```
AD-A100 429    DAYTON UNIV  OH SCHOOL OF ENGINEERING                    F/G 9/2
               COMPUTER PROGRAM DOCUMENTATION FOR MICROCOMPUTER IMPLEMENTATION--ETC(U)
               OCT 78  F G ALBERS, J CROUCH                      F33657-77-C-0477
UNCLASSIFIED   UDR-TR-78-98                                             NL
```

1 OF 2
AD
A100429

UDR-TR-78-98
UDSE-78-09

COMPUTER PROGRAM DOCUMENTATION FOR MICROCOMPUTER IMPLEMENTATION STUDY

Dr. F. Gerard Albers
Dr. J. Crouch

School of Engineering
University of Dayton
Dayton, Ohio 45469

OCTOBER 1978

Final Report June 1977 - September 1978

AERONAUTICAL SYSTEMS DIVISION
WRIGHT-PATTERSON AIR FORCE BASE
OHIO 45433

(14) UDR-TR-78-98
UDSE-78-09

(1)

COMPUTER PROGRAM DOCUMENTATION FOR MICROCOMPUTER IMPLEMENTATION STUDY.

(10)

[redacted] F. Gerard Albers
[redacted] J. Crouch

(12) 449

School of Engineering
University of Dayton
Dayton, Ohio 45469

F33657-77-C-0477 new
SC

(11) OCTOBER 1978

(15) F33657-77-C-0477

(9) Final Report, June 1977 - September 1978

AERONAUTICAL SYSTEMS DIVISION
WRIGHT-PATTERSON AIR FORCE BASE
OHIO 45433

4 1377 .V

TABLE OF CONTENTS

SECTION PAGE

SECTION PAGE

APPENDIX

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS[*]

a_x	x-directed airplane acceleration (ft/sec^2)
a_z	z-directed airplane acceleration (ft/sec^2)
B	distance from airplane C. G. to bobweight rotational axis. See Figures 2. 2 and 2. 3.
$\bar{c}$	bobweight moment arm. See Figure 2. 2.
D_c	stretch viscous damping coefficient
D_1, D_2	viscous damping coefficients
F_A	actuator force
f_1, f_2	coulomb friction coefficients
F_B	bobweight force
F_F	spring feel force
F_p	pilot force
F_s	control loading force
FF	intermediate actuator stick force variable
g	acceleration of gravity (ft/sec^2)
G	transfer function
G_E, G_A, G_R	control gear ratios
i_v	electrohydraulic valve current
I_x, I_y, I_z, I_{xz}	airplane moments
L	rolling moment/I_x (sec^{-2})
M	pitching moment/I_y (sec^{-2}). Also used for mass of arm. See Figure 2. 3.

[*] Differentation with respect to time is indicated by a dot. Thus $\dot{u} = \frac{du}{dt}$.

M_1, M_2	control system masses
m	airplane mass. Also used for bobweight mass.
N	yawing moment/I_z (sec^{-2})
N_y	y-directed dynamic airplane acceleration in g units
N_z	z-directed dynamic airplane acceleration in g units
p	roll rate (degrees per sec)
q	pitch rate (rad/sec or ^{o}/sec)
R	pilot force moment arm. See Figures 2.2 and 2.3. Also used as sample rate (R = 1/T).
r	yaw angle rate (degrees per sec)
s	Laplace variable
T	sample interval
u	x-velocity perturbation (ft/sec)
U, U_o	undisturbed airplane x velocity (ft/sec)
v	y-velocity perturbation (ft/sec)
w	bobweight weight
x_{sp}	valve spool position
X	x-force component/mass (ft/sec^2)
X_a, X_{a_1}, X_{a_2}	control stick grip roll system displacements (right plus)
X_e, X_{e_1}, X_{e_2}	control stick grip pitch system displacements (aft plus)
X_r, X_{r_1}, X_{r_2}	rudder pedal system displacements (right foot forward plus)
X_u, M_u, etc.	dimensional stability derivatives defined as: $\lim\limits_{u \to 0} \frac{\partial X}{\partial u}$, $\lim\limits_{u \to 0} \frac{\partial M}{\partial u}$, etc.
x_n	value of x at $t = n \cdot T$
Y	side force component/mass (ft/sec^2)
y_{n-1}	value of y at $t = (n-1)\,T$
Z	z-force component/mass (ft/sec^2)

α	airplane perturbation angle of attack (w/U_o radians)
δ	control surface angle
$\bar{\delta}$	command control surface angle
δ_a	aileron angle (right up positive, degrees)
δ_e	elevator angle (T. E. down positive, degrees)
δ_r	rudder angle (left positive, degrees)
Δ	delay operator (i. e. $\Delta y_n = y_{n-1}$)
ψ	azimuth angle (degrees)
ϕ	roll angle (degrees)
θ	airplane pitch angle (radians or degrees)
θ_n	value of θ at $t = n \cdot T$
ℓ	length of limb bobweight arm. See Figure 2.3.
F_{Br}	breakout force level

CROSS REFERENCE TABLE

	Math Symbol	Mnemonic	Description
Pitch	X_P	PITCH	Stick Displacement - Pitch Axis
	$\dot{X}_P$	PDOT	Stick Velocity - Pitch Axis
	K_P	KPLIMF	Pitch Limit Factor
	K_{L_P}	KPLIMLEAD	Pitch Limit Lead Factor
	K_{B_P}	KPBRKLEAD	Pitch Breakout Lead Factor
	N_Z	NZ	Airframe Normal Acceleration - Pitch Axis
	$\dot{q}$	QDOT	Airframe Angular Acceleration - Pitch Axis
	0.64	KPVISC	Pitch Viscous Damping Coefficient
	1.5	KPCOUL	Pitch Coulomb Friction Coefficient
	2.5	KPBRAK	Pitch Breakout Force Coefficient
	3.0	KNZ	Pitch Normal Acceleration Coefficient
	0.026	KQDOT	Pitch Acceleration Coefficient
Roll	X_R	ROLL	Stick Displacement - Roll Axis
	$\dot{X}_R$	RDOT	Stick Velocity - Roll Axis
	K_R	KRLIMF	Roll Limit Factor
	K_{L_R}	KRLIMLEAD	Roll Limit Lead Factor
	K_{B_R}	KRBRKLEAD	Roll Breakout Lead Factor
	N_Y	NY	Airframe Normal Acceleration - Roll Axis
	$\dot{p}$	PPDOT	Airframe Angular Acceleration - Roll Axis
	4.5	KRSPRING	Roll Spring Coefficient
	0.06	KRVISC	Roll Viscous Dumping Coefficient
	1.5	KRCOUL	Roll Coulomb Friction Coefficient
	2.0	KRBRAK	Roll Breakout Force Coefficient
	5.0	KNY	Roll Normal Acceleration Coefficient
	0.008	KPPDOT	Roll Acceleration Coefficient
Yaw	X_Y	YAW	Rudder Pedal Displacement - Yaw Axis
	$\dot{X}_Y$	YDOT	Rudder Pedal Velocity - Yaw Axis

Math Symbol		
K_Y	KYLIMF	Yaw Limit Factor
K_{L_Y}	KYLIMLEAD	Yaw Limit Lead Factor
K_{B_Y}	KYBRKLEAD	Yaw Breakout Lead Factor
15.0	KYSPRING	Yaw Spring Coefficient
0.905	KYVISC	Yaw Viscous Dumping Coefficient
4.5	KYCOUL	Pay Coulomb Friction Ceofficient
5.0	KYBRAK	Yaw Breakout Force Coefficient

SECTION 1
INTRODUCTION

The advent of the microcomputer has revolutionized the field of computer technology in much the same way that the transistor revolutionized the field of electronics design. Simulators have characteristically been designed around large, high speed, general purpose computers. The microcomputer, on the other hand, is usually employed as a stand-alone, dedicated subsystem component. The concepts of the stored program, "fetch and execute" architecture are common between the two but the similarity does not go much further. For example, the memory of the microcomputer is distinctively divided into program instructions and data storage. Instructions are stored in an unalterable read only memory (ROM or PROM) and variable data is stored in random access memory (RAM). Thus, there are no requirements for loading the program and there are no memory overlays. The software is neatly arranged into one functioning "module" although several software packages (also called modules) may have been linked together to form the real-time "modules." There are no operating systems or executives for the microcomputer in the same sense as for its larger brother. Furthermore, (and this is probably the most significant point), the microcomputer "software" is very hardware oriented. The liberalizations and generalizations concerning input/output, memory access, and mathematical operations which can be used with the larger systems simply do not exist for the micro. The total microcomputer system design requires that the hardware not be separated from the software; the designer must thoroughly understand both. Similarly, a discussion of the software must include at least a relevent discussion of the hardware.

This introduction is meant to aid the reader in analyzing the approach which was taken in developing this document. In many cases the words, terminology and even general concepts of DI-H-3277/M4 do not apply to the microcomputer. On the other hand, some topics (such as hardware related material) has been included although not called for in the DID. As one might suspect from the foregoing discussion, the title "Computer Program Documentation" does not completely describe the true nature of this document. However, the following sections are developed to correspond as closely as possible with sections required by DI-H-3277/M4.

SECTION 2
SOFTWARE ELEMENT FAMILY TREE

The entire control loader software was written in the Intel high-order microcomputer language called PLM (PL/M-80), with the exception of the analog-to-digital conversion routine which was programmed in assembly language. The programs were structured into a utility module, a scan module, and a simulator module. As the name implies the utility module provided system support in the form of calibration and test routines, system interface procedures, and system initialization. The scan module causes twelve of the available sixteen A-to-D converter channels to be scanned sequentially and the converted values to be placed in the appropriate RAM locations for later use. The simulator module contains the frame rate generator, simulator initialization routines, and the simulator equations themselves. These three modules were independently compiled and linked into machine code. The executable machine code was placed in PROM (programmable read only memory). The variable data and equation coefficients are placed in RAM (Random Access Memory) during real-time.

The software is best described by structuring it into elements as illustrated in Figure 2. 1. The elements that reside in the utility module are designated by a "U"; the simulator elements by an "S". The scan module is shown as a separate entity.

The "main program" is entered whenever a reset occurs. It initializes all system parameters and immediately calls the simulator into action. The executive is actually part of the main program and it administers to the six programs (0 through 5) of the microcomputer. When a program is terminated, control returns to the executive. Since the simulator is one of these programs, an exit from it will pass control to the executive regardless of how it was entered. The scan module is only used by the simulator for analog input purposes. In addition, the simulator contains its own routine for changing the equation coefficients and constants.

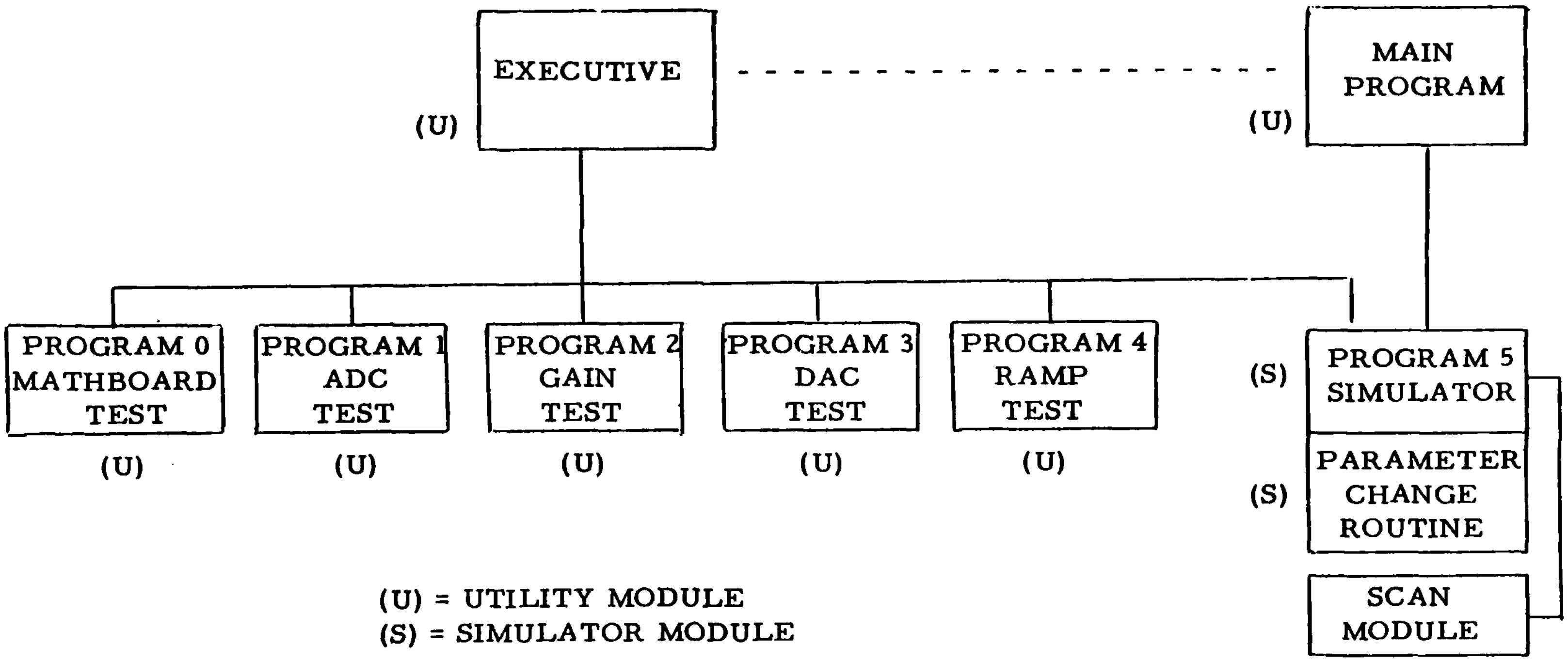


Figure 2.1 Software Elements Family Tree

SECTION 3

TRAINING PROGRAM OPERATION OVERVIEW

This section is designed to provide an overview of the software system structure and event sequencing, memory allocation and simulation framing control. Due to the nature of the microcomputer the software requires no loading, nor an operating system for control. Therefore, only overview items which are relevent to the microcomputer system are discussed.

3.1 SOFTWARE SYSTEM STRUCTURE

System control always resides in either the executive (see Figure 2.1) or in the simulator. The simulator is simply an "infinite loop" of statements which continuously (120 times per second) solves the control loading equations. It accepts data inputs through the analog input interface (and scan module) and controls the force on the stick through the analog output interface. The only way to exit the simulator is through the parameter change routine in which case control passes to the executive. The executive queries the system operator (through the teletype) for a program number which it executes immediately. When a program is exited (including the simulator) control passes once again to the executive. The teletype also plays a roll in the parameter change routine in that the operator specifies which parameter is to be changed and to what value. The operator can also select whether to return to the simulation or to pass control to the executive at this time.

3.2 MEMORY ALLOCATION

The microcomputer system is equipped to accomodate 8K (four 2716's) of programmable read only memory (PROM) and 2K (eight 2113's) of random access memory (RAM). The three modules (utility, simulator and scan) were independently compiled and linked into a single object module which was placed in the PROM. Certain PLM library routines were linked into the object module also. The exact locations of the modules and routines in PROM is relatively unimportant since none of the PROM locations are alterable. However, this type of detailed information can be gotten in the cross-reference listing and the link and locate maps (see Volume 3). At this point it is sufficient to say that the PROM memory lies in the lowest area of memory since execution starts at location 0 (see Figure 3.1). Presently 6820 (decimal) locations of the PROM's are actually used out of the available 8096. Variables are stored in RAM which starts at 3800 H (hex). Only 500 of the available 2000 locations are used. The upper portion of memory is reserved for the peripheral circuit boards in a memory mapped I/O configuration. The analog-to-digital converter (ADC) has a base address at F700H, the digital-to-analog converter (DAC) at F708H, and the high speed mathboard at FFF0H.

3.3 COMPILING, LINKING AND LOCATING

The three modules involved in this simulator were individually compiled or assembled into executable machine code. Each module's machine code then had to be linked together so that they could interact as a unit. This single block of machine code and the address references within it had to be properly located so that it performed properly in the System 80/20. All of these functions were performed using the services of the Intel ISIS-II operating system as described in the following paragraphs.

3.3.1 Compiling

The Utility and Simulator Modules were written in PL/M-80, Intel's high order langugae. The compiler is called PLM80 and it is invoked through ISIS by executing PLM80 and specifying the source code

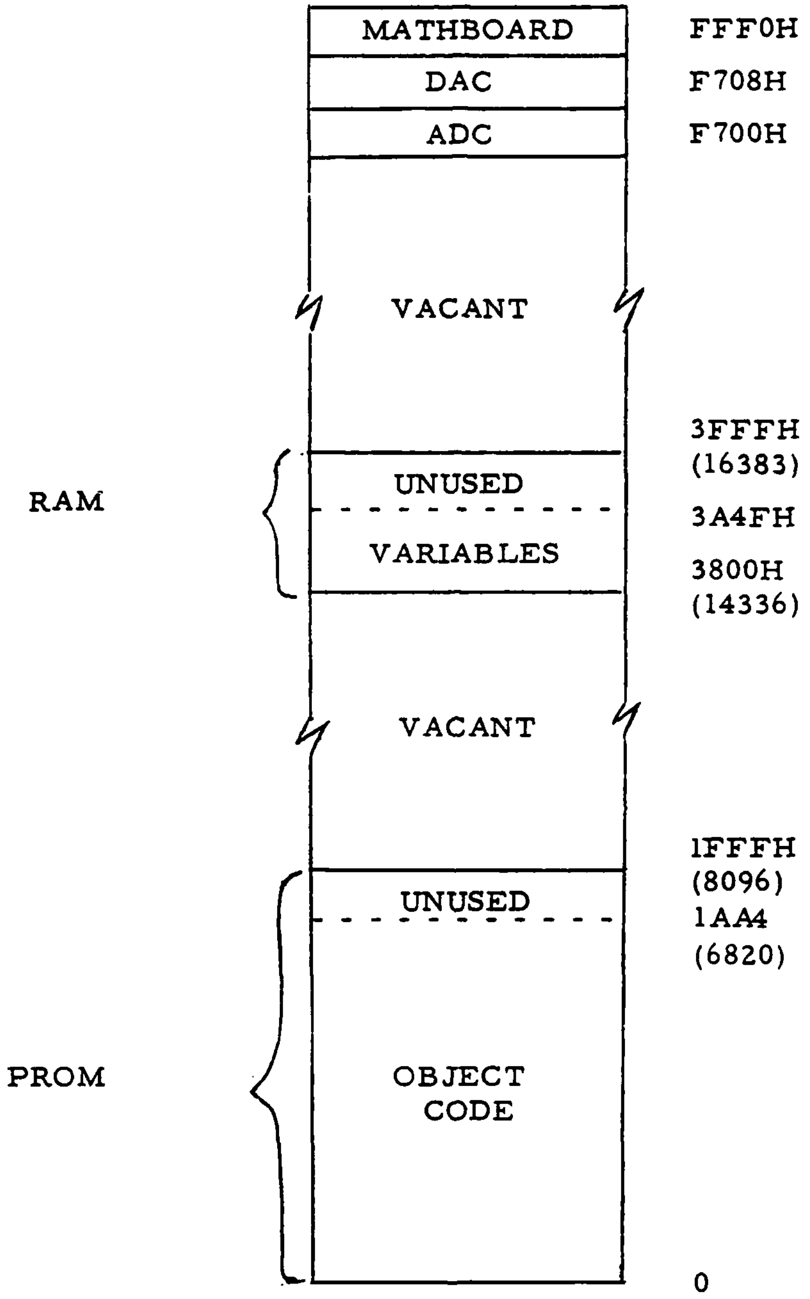


Figure 3. 1 Memory Allocation

file name and the desired compiler options. The compiler commands for the Utility (CNTU6.SRC) and Simulator (CNTS85.SRC) modules were respectively:

```
PLM80  :F1:CNTU6.SRC    SYMBOLS DEBUG XREF PAGEWIDTH (132)
PLM80  :F1:CNTS85.SRC   SYMBOLS DEBUG XREF PAGEWIDTH (132)
```

The :F1: specifies that at the time of compilation the source file resided on disk drive number 1. An explanation of the options and other features of the compiler can be obtained in the ISIS-II PL/M-80 Compiler Operator's Manual. As a result of executing the above commands, two object files are created on disk drive 1 with the names CNTU6.OBJ and CNTS85.OBJ respectively.

Similarly, the Scan Module source SCAN.SRC is assembled into object code by executing the following ISIS command:

```
ASM80  :F1:SCAN.SRC
```

The optional compiler controls were incorporated with the source listing and an explanation of these can be obtained from the ISIS-II 8080/8085 Macro Assembler Operator's Manual. This assembler likewise produces an object file with the same generic name as the source; namely, SCAN.OBJ.

3.3.2 Linking

The three object files must now be linked together into one unit. In addition, certain routines from the PLM library are required as a result of using the PLM compiler. The task of unifying these four files into one unit of concise machine code is the job of the ISIS-II linker called LINK. A description of this program can be obtained from the ISIS-II System User's Guide. The actual link command for this project can be obtained from the link listing. It merely specifies that LINK is to link :F1:CNTU6.OBJ, :F1:CNTS85.OBJ, :F1:SCAN.OBJ, and PLM80.L1B to a new file called CNT85.LNK on disk drive number 1.

3.3.3 Locating

The object code file for the system now resides in a disk file called CNT85.LNK. This file must be adjusted so that the executable code, the variables and other factors lie in the proper location. This is the task of program LOCATE which is also described in the ISIS-II System User's Guide. The exact command used to invoke this program can be obtained from the locator listing. This listing contains the final addresses of all the system parameters including the address of the starting location of the code for each line in the PLM listings. The final object code for the entire system is located in a file called CNT85. It is this code which was programmed into the system PROM's.

3.4 SIMULATION FRAMING CONTROL

Program execution starts from location 0 which is in the "main" program. A series of initializations are performed and control passes to the simulator. The simulator, in turn, initializes certain parameters and functions, one of which is the frame rate generator. This generator is a procedure which loads a hardware counter. This counter is decremented by the system clock while the simulator is calculating the control loading values. When all three control axes have been calculated and the values have been written to the DAC's, the simulator waits for the counter to decrement to zero which fires an interrupt (at level 1). This interrupt re-initializes the frame rate generator and allows control to pass back to the beginning of the simulator loop. Thus, it can be seen that the frame time available for computation is held constant 8.33 milliseconds (1/120 second) while the actual time consumed in computation will vary according to the processing requirements of each axis. The worst-case (maximum) amount of time for all computation is approximately 7.8 milliseconds. This maximum occurs when all three actuators are in the linear portion of their ranges. Figure 3.2 illustrates the relative times consumed for scanning the analog input channels and for the calculations associated with each axis.

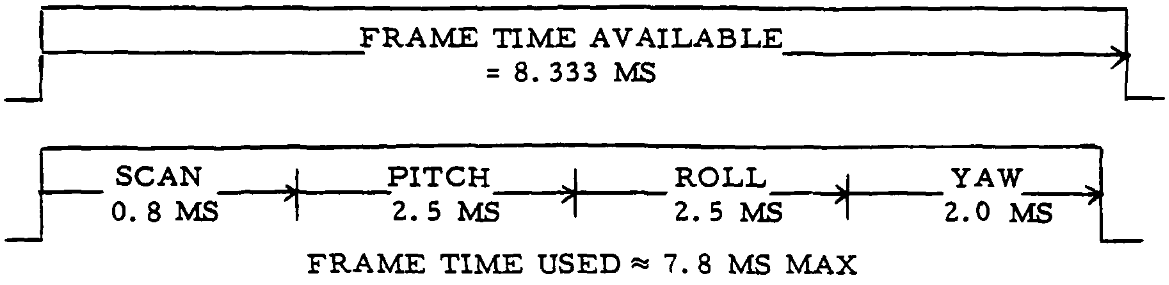


Figure 3.2 Frame Timing

SECTION 4
CONTROL LOADING FORCES AND EQUATIONS

4.1 SIGN CONVENTIONS

Since the McFadden Control Loader was used in the simulator, we elected to adapt our sign conventions to it. Table 4.1 summarizes the maximum stick parameters and the sign conventions of the McFadden control loader and, thus, of the overall system. The stick and rudder pedal displacements produced one volt for every inch of travel at the McFadden interface even though the mechanical limits prohibited travel over a full 10 inches (corresponding to the analog full scale of 10 volts). All other parameters are scaled to 10 volts at their maximum value. The displacement and velocity parameters are obviously control loader outputs (computer inputs) and the force is a control loader input (computer output). Thus, if the stick is moved aft (positive direction, positive voltage output) an opposing force is produced by the computer which places a negative voltage input to the control loader resulting in a forward force.

TABLE 4.1 SIGN CONVENTIONS

Axis	Parameter	Positive Direction	Max Value
Pitch	Displacement	Aft	7 in (5 in forward)
	Velocity	Aft	50 in/sec
	Force	Aft	150 pounds
Roll	Displacement	Right	7 inches
	Velocity	Right	60 in/sec
	Force	Right	100 pounds
Yaw	Displacement	Right Rudder	3.25 inches
	Velocity	Right Rudder	50 in/sec
	Force	Right Rudder	200 pounds

Thus, if we select positive directions for control motion, velocity, and force to be aft stick, right stick and right pedal forward, the output feel force will properly oppose the pilot action.

The control loading forces are functions of control position and velocity. In the cases of the pitch and roll axes, we base all forces on the motion of the top of the control stick in inches and inches per second. Control stick aftward is positive displacement (X_e) and positive velocity ($\dot{X}_e$). Thus, positive X_e results in upward elevator trailing edge (negative elevator angle, δ_e) and the aircraft moves toward a nose up attitude. Positive pilot force (F_p) is aftward while the resulting control loading force (F_s) is forward (negative) so that the net force on the control stick is (F_p+F_s) in the positive X_e direction. Positive displacements, velocities, control angles and forces are illustrated in Figure 4.1

Control stick right giving right aileron trailing edge up is positive X_a, $\dot{X}_a$, and δ_a. The aircraft moves toward right wing low. Positive forces are shown in Figure 4.1

In the case of the yaw axis, positive displacement (X_r) and velocity ($\dot{X}_r$) is right rudder pedal forward. This gives right rudder deflection (δ_r) which is negative and the aircraft moves toward nose right. Positive pilot force is forward on the right pedal while negative force is forward on the left pedal.

4.2 LOADING FORCES DUE TO CONTROL VELOCITY AND POSITION

In an airplane without power assist the opposing force the pilot feels is due to mechanical stops or friction in the control members plus the hinge moment which must be overcome to hold the control surface in place. Due to the magnitude of these forces in high speed flight, many modern aircraft have fully powered control systems in which an artificial feel system provides the pilot a feedback force approximately

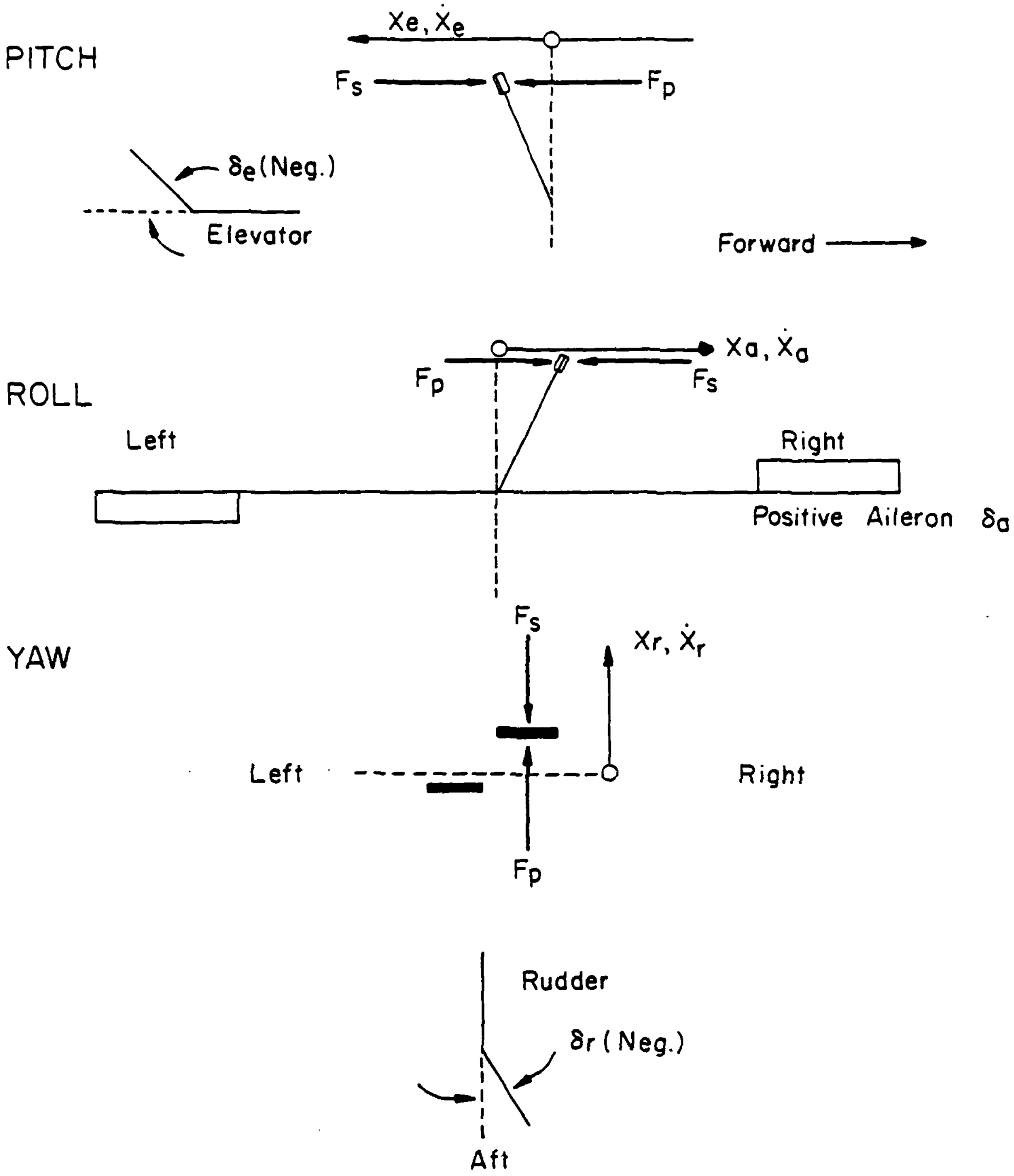


Figure 4.1 Sign Conventions

proportional to the force that would be required without power. Sometimes the force may be modified by adding artificial damping, for example. In the simulator, this control loading (feel) force must approximate, as nearly as possible, the actual feel force provided in the real airplane.

The simulator control loading force is formed by adding together various components, each of which embodies a characteristic observed in the actual airplane control loading force. The components of loading force included in the control loading microcomputer are described in Table 4.2

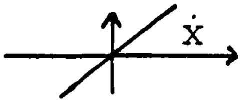
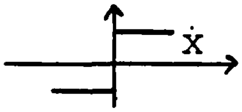
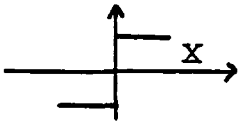
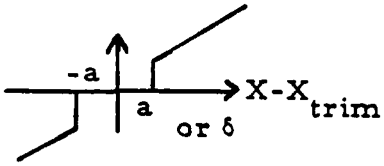
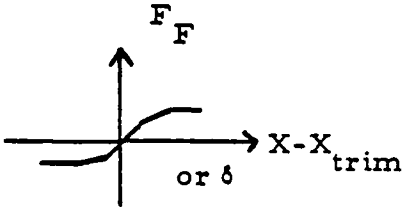
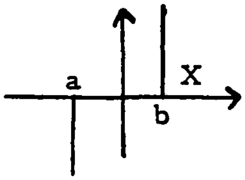
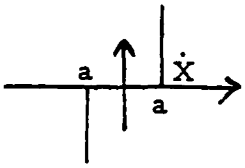
The components shown in Table 4.2 are common to many mechanical systems. The viscous friction term is the linear part of the frictional force and it appears partially due to the motion of the control surface in the fluid (air). However a certain level of viscous friction is desirable for stability and feel quality. The A-10 aircraft, for example, has an eddy current damper included as part of the control system to increase the viscous damping.

The coulomb friction term simulates the dry friction resulting from cables, pulleys, and arms which necessarily have sliding contact with each other and with other members in the real airplane. This frictional force is of fixed magnitude always opposing the motion of the control.

The breakout force is that force which must be applied to get any mechanical system off of "dead center." It is similar to lifting a weight from a table. The applied force must build up to at least equal the weight force before the weight will move. Likewise, in causing a pulley wheel to turn, the applied torque must build up to equal the load torque before the wheel will turn.

Deadband represents the freeplay in a mechanical system around the trim or equilibrium position of the control. The freeplay is caused by such things as cable slack and loose fittings. The deadband may

TABLE 4.2 CONTROL LOADING COMPONENTS

Name	Characteristic	Mathematical Designation
Viscous Friction		$k\dot{X}$
Coulomb Friction (Dry or Sliding Friction)		k sign $(\dot{X})$ or $k(\dot{X}/\lvert\dot{X}\rvert)$
Breakout (Preload)		k sign $(X-X_{trim})$
Deadband		$[u(\delta-a)+u(-\delta-a)]F(\delta)$ where $u(\delta)$ is the unit step function and where $\delta = X-X_{trim}$
Feel Spring		$k_i(\delta-\delta_i)+b_i$ continuous, but with discontinuous slope. Here $\delta = X-X_{trim}$
Position Limiter		$k(X-b)$ if $X \geq b$ 0. for $-a<X<b$ $k(X+a)$ if $X \leq -a$
Velocity Limiter		$k(\dot{X}-a)$; $\dot{X} \geq a$ $k(\dot{X}+a)$; $\dot{X} \leq -a$

exist either right at the cockpit control or elsewhere in the system such as at the elevator actuator in the tail. Actually, deadband exists throughout the system at every interface between members.

The feel spring force represents the hinge moment of the control surface. Since most modern aircraft have fully powered control systems, a feel spring is included to artificially provide the pilot a force suitably like that due to hinge moment. In the aircraft simulator, the feel spring force is made as nearly like the aircraft feel spring as possible. Hinge moment varies with control surface deflection, angle of attack, trim angle and dynamic pressure so the feel spring may reflect all of these variations. Usually, the feel spring force is a nonlinear function of control deflection which may be made dependent on dynamic pressure.

The position limit force represents the physical limit on motion of the control. When the position limit is reached, the pilot must feel a realistic hard stop just like the hard stop he feels in the aircraft. The feel of the stop is better if there is a high gradient of force build-up rather than the sudden application of a high force. Also, the high frequency content of the force signal is less.

Aircraft control systems usually impose limiting velocities of motion of the control stick and rudder pedals. These result from nonlinearities in the various system elements or they may be deliberately imposed. Limiting velocities are desirable in that they protect the system components from damage. Like the position limiting force, the velocity limiter is depicted as a high force build-up when the velocity exceeds the limit.

4.3 BOBWEIGHT FORCES

The inertial effects of the aircraft accelerations upon the parts of the control system affect the control loading forces. These inertial forces, referred to as bobweight forces, may be calculated when the airframe motion variables are known. Perhaps the greatest contributions to the bobweight forces come from the control surfaces themselves except when the control system is fully powered. In that case, the aircraft control system may include an actual bobweight (as in the A-10) to provide an inertial stick feel force. A bobweight may sometimes be used only in the pitch axis (A-10). If the aircraft control system has bobweight forces, these should be present in the simulator control loading system also.

Consider an idealized bobweight attached to the airplane as shown in Figure 4.2. The bobweight obviously will exert an inertial moment on the control stick. We assume the motions are restricted to the plane of the paper. Then the dynamic* acceleration acting at the CG of the airplane is:

$$a_z = (\dot{w} - U_o \dot{\theta}) = U_o(\dot{\alpha} - q) \qquad \text{(Positive downward)} \qquad (4.1)$$

In g-units, with positive g's meaning upward acceleration, we have:

$$N_z = -\frac{a_z}{g} = -\frac{U_o}{g}(\dot{\alpha} - q) \qquad (4.2)$$

Thus, we see how angle-of-attack rate and pitch velocity lead to z-directed acceleration. Positive G-acceleration will result in the bobweight trying to rotate downward and a forward stick force will be felt by the pilot. If the bobweight has mass, m, and weight, W, we have:

$$F_s = F_p = -(m \cdot a_z) \cdot \bar{c}/R \qquad (4.3)$$

$$F_s = -\frac{W}{g} \cdot (-N_z \cdot g) \cdot \bar{c}/R = W \frac{\bar{c}}{R} N_z$$

Symbols defined in Figure 4.2.

*dynamic z-acceleration is total z-acceleration minus 1.

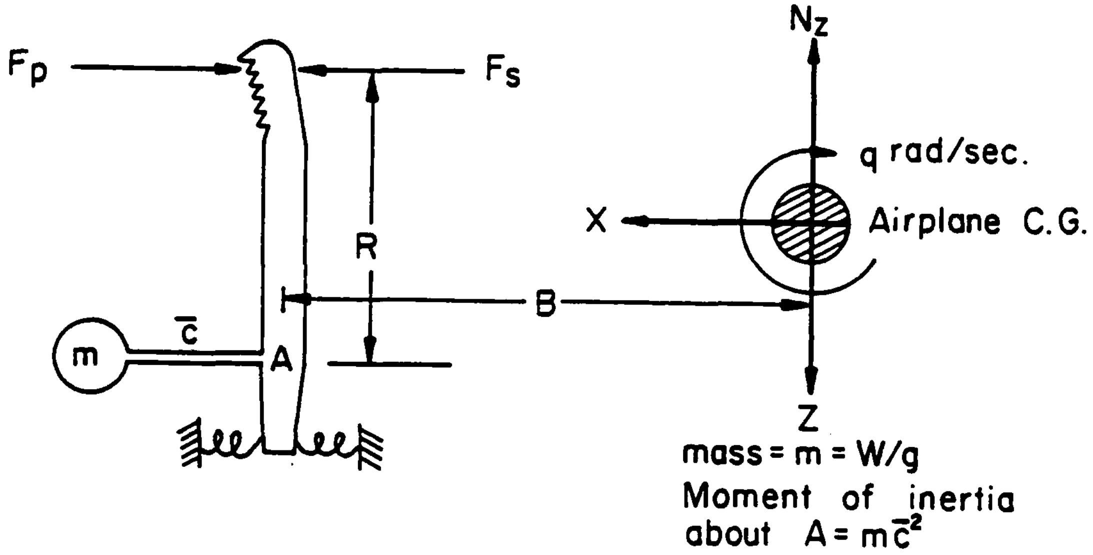


Figure 4.2 Idealized Bobweight in Pitch Axis

If the airplane experiences an angular acceleration about its CG (q), the bobweight will lag behind. For a positive $\dot{q}$ (pitch up), the bobweight will try to rotate downward creating a forward stick force as shown:

$$F_s = F_p = \frac{m(\bar{c}+B)\dot{q}\bar{c}}{57.3R} = \left(\frac{W(\bar{c}+B)\bar{c}}{57.3gR}\right)\dot{q} \qquad (4.4)$$

where $\dot{q}$ is in degrees per second per second.

The total bobweight feel force is the sum:

$$F_s = \left(W\frac{\bar{c}}{R}\right) N_z + \left(\frac{W\bar{c}(\bar{c}+B)}{57.3gR}\right)\dot{q} \qquad (4.5)$$

If $R = 1.67$ ft, $W = 6.26$ lb., $\bar{c} = .8$ ft, and $B = 15.31$ ft we find:

$$F_s = 3\ N_z + .0262\ \dot{q} \qquad (4.6)$$

as on the A-10.

4.4 LIMB BOBWEIGHT FORCES

In the presence of aircraft accelerations, the pilots arm and hand gripping the control stick will behave like a bobweight. The limb will try to lag behind the inertial airplane accelerations and the pilot must exert forces to hold the stick/limb combination steady.

In vertical accelerations and pitch accelerations, the limb bobweight forces on the stick would appear to be mainly in the z-direction (vertical) and we have no good way to impart simulated vertical stick feel forces to the pilot. However, in side accelerations and roll accelerations, the limb bobweight forces on the stick would be y-directed (sideward) and we can impart simulated side stick feel forces to the pilot.

Consider an idealized bobweight, as shown in Figure 4.3 simulating the pilots arm/hand acting on the control stick. We make the following simplifying assumptions:

1) Pilot's body immobile except for arm.

2) Mass of arm, M, concentrated at a point "ℓ" distant from body.

3) Action of acceleration on arm simulated by stick force, F_s, acting distance "R" from shoulder.

4) Arm a distance "B" above x-axis.

5) The only forces being considered are those due to side acceleration (N_y) and roll acceleration ($\dot{p}$).

6) Arm motion limited to rotation about vertical axis except as restrained by control stick.

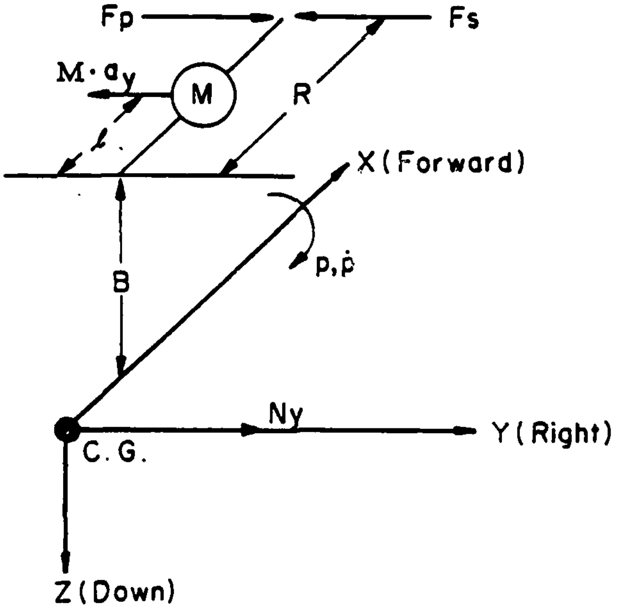


Figure 4.3 Limb Bobweight Simulation

Then the inertial moment on the arm (M) due to side acceleration, N_y, is such as to make the arm move to the left. We simulate this moment by a stick force, F_s, as shown, which produces the same moment:

$$F_s \cdot R = (M\, a_y) \cdot \ell \tag{4.7}$$

$$F_s = Mg \left(\frac{a_y}{g} \right) \frac{\ell}{R} \tag{4.8}$$

$$F_s = Mg \frac{\ell}{R} N_y \tag{4.9}$$

Here, a_y is y-directed acceleration and $\frac{a_y}{g}$ is N_y in G-units. This stick force and moment will require the pilot to supply an equal and opposite force/moment to keep the stick steady. Thus he is required to supply the same counter-acceleration moment he would supply in flight with a side load.

The inertial moment on the arm (M) due to roll acceleration ($\dot{p}$) is such that the arm would tend to rotate to the left unless the pilot applies a counteracting moment. We simulate the inertial moment by a stick force F_s, as shown, which produces the same moment:

$$F_s \cdot R = (M\, a_y) \cdot \ell \tag{4.10}$$

where $a_y = \dot{p}B$

Then

$$F_s = \left(\frac{MB\ell}{R} \right) \dot{p} \tag{4.11}$$

If $\dot{p}$ is in degrees per second per second then:

$$F_s = \left(\frac{MB\ell}{57.3 \cdot R} \right) \dot{p} \tag{4.12}$$

As before, this stick force will require the pilot to supply the proper counteracting moment as he would in actual flight with roll acceleration.

The total limb bobweight stick feel force is the sum of that due to side acceleration and that due to roll acceleration:

$$F_s = \left(Mg \frac{\ell}{R} \right) N_y + \left(\frac{MB\ell}{57.3 \cdot R} \right) \dot{p} \qquad (4.13)$$

If we take some reasonable values for the parameters:

$Mg = 10$ lbs.

$\ell = 8''$ (.67 ft)

$R = 1.33$ ft

$B = 3$ ft

We then have

$$F_s = 5 N_y + .008 \dot{p} \qquad (4.14)$$

This is the expression used for limb bobweight force in the present program.

4.5 CONTROL LOADING EQUATIONS

Adding together all of the sources of control loading force discussed in sections 4.2, 4.3, and 4.4, we now obtain the loading force for each axis.

The microcomputer task is to calculate the stick (or pedal) feel force to be input to the force control loop and generated by the control loading actuator. Information on the A-10 furnished by the Air Force was for a two-degree-of-freedom system for each axis (see Appendix A). We have consolidated each axis as a single degree-of-freedom system as follows:

4.5.1 Pitch

$$F_s = \underset{\substack{\text{viscous}\\\text{friction}}}{.64\,\dot{X}_e} + \underset{\text{coulomb friction}}{1.5\ \text{sign}\,(\dot{X}_e)} + \underset{\text{breakout}}{2.5\ \text{sign}\,(X_e - X_{e_{trim}})}$$

$$+ \underset{\text{bobweight}}{3.\,N_z + .0262\,\dot{q}} + \underset{\text{feel spring}}{F_F \lambda_p} + \underset{\substack{(-6.7\text{ in. aft,}\\+2.7\text{ in. fwd.})}}{F_{\text{travel lim.}}}$$

$$+ \underset{(\pm 5.236\text{ in/sec})}{F_{\text{vel. lim.}}} \qquad (4.15)$$

X_e = position of stick-grip (fore & aft), inches. (Positive aft.)

$X_{e_{trim}}$ = integrated output of on-off trim switch.

N_z = dynamic normal acceleration.

$\dot{q}$ = pitch acceleration, degrees per second per second.

F_F = a piecewise linear function of $(X_e - X_{e_{trim}})$ with break points as shown in Table 4.3.

λ_p = a function dependent on hinge moment. It is primarily a magnification factor sensitive to dynamic pressure. (λp = 1 in present program.)

4.5.2 Roll

$$F_s = \underset{\substack{\text{viscous}\\\text{friction}}}{.06\,\dot{X}_a} + \underset{\text{coulomb friction}}{1.5\ \text{sign}\,(\dot{X}_a)} + \underset{\text{breakout}}{2\ \text{sign}\,(X_a - X_{a_{trim}})}$$

$$+ \underset{\text{feel spring}}{4.5\,(X_a - X_{a_{trim}})\lambda_p} + \underset{\text{limb bobweight}}{5\,N_y + .008\dot{p}} \qquad (4.16)$$

TABLE 4.3 PITCH FEEL SPRING BREAK POINTS

$(X_e - X_{e_{trim}})$ inches	F_F pounds
(forward)	
-2.7	-12.25
-1.5	-7.5
0.	0.
1.3	7.
3.5	12.25
5.7	16.0
(aft)	

$$+ F_{\text{travel lim.}} \quad + \quad F_{\text{vel. lim.}}$$

(±3.1 inch) (±6.5 in/sec)

X_a = lateral position of stick-grip, inches (positive right).

$X_{a_{trim}}$ = integrated output of on-off trim switch.

N_y = lateral G-force.

$\dot{p}$ = roll acceleration, degrees/sec/sec.

λ_p = magnification factor dependent on hinge moment (λ_p = 1. in present program).

No apparent deadband in roll for A-10.

4.5.3 Yaw

$$F_s = \underbrace{.905\,\dot{X}_r}_{\text{viscous friction}} + \underbrace{4.25 \text{ sign } (\dot{X}_r)}_{\text{coulomb friction}} + \underbrace{5. \text{ sign } (X_r)}_{\text{breakout}} \tag{4.17}$$

$$+ \underbrace{15.\, X_r \lambda_p}_{\text{feel spring}} + \underbrace{F_{\text{travel lim.}}}_{(\pm 3.5 \text{ in})} + \underbrace{F_{\text{vel. lim.}}}_{(\pm 11.9 \text{ in/sec})}$$

No rudder trim shown on A-10 data.

No deadband shown.

X_r = rudder pedal position, inches (left pedal fwd. is positive).

λ_p = magnification factor dependent on hinge moment (λ_p = 1. in present program).

4.6 CONTROL LOADING LOOP

The University elected to attempt to simulate the entire control loading loop (pitch axis only) to gain confidence in our understanding of the system prior to actually wiring up the hardware. The basic part of the loop is a two-stage, electrohydraulic valve/actuator which generates the feel force for the pilot to sense. Reference 4 discusses this type of valve. Figure 4.4 is a block diagram of what we feel is a reasonable representation of the loop. Note that the expression for actuator torque is nonlinear so the responses were obtained by an iterative technique.

Throughout the system estimates were made of time constants and dimensions. The step response of this system is included in Section 5.2 of this report. This response was obtained by simulating the control loop on the University's computer. We did not use the airframe equations in this simulation so airframe variables such as dynamic pressure, N_z, N_y, q, p, and trim were not available. Also travel limits and velocity limits were not used.

We begin a brief description of the control loop operation at the right side of Figure 4.4 where the pilot force is input to the control stick. The difference between the pilot force and the opposing actuator force (F_A) causes acceleration of the stick grip leading to stick velocity ($\dot{X}_e$) and position (X_e). $\dot{X}_e$, being the velocity of the stick, determines the flow rate (Q) through the hydraulic valve.

Stick position and velocity are furnished to the microcomputer for use in generating the command stick feel force (F_s). The test results shown in Section 5 included only viscous friction, breakout force, and feel spring force among the components of F_s.

The command force, F_s, is next compared to the actual actuator force, F_A, giving an error signal (F_s-F_A). The error voltage leads to valve current, i_v, and valve spool position, x_{sp}, through appropriate transfer functions as shown.

The actuator steady state torque is a nonlinear function of valve current, spool position, and flow rate. The steady state actuator generated stick force, FF, is calculated assuming a control stick length of 20 inches. Since steady state conditions are not arrived at immediately, a time delay smoothing transfer function converts FF to F_A which is the actual actuator force.

The nonlinearity in the torque expression necessitated use of an iteration procedure to obtain convergence for each update of actuator feel force. The step response shown in Section 5.2 was for an input ten (10) pound pilot step force. The response is seen to be very fast as it must be to be realistic. The force overshoot seen after 9/120 second is necessary to decelerate the stick and reach an eventual constant stick displacement.

Constants for Figure 4.4 (assumed)

K_1	=	.02 inches/milleamp
T_1	=	.0002 sec^{-1}
K_3	=	2. inches^3
P_s	=	1500. psi. (supply pressure)
C_z	=	20. $\text{in}^3/\text{sec}/(\text{pound})^{1/2}$
C_f	=	.1 inches^2
T_2	=	.01 sec^{-1}
m	=	.0222 pound sec^2/inch

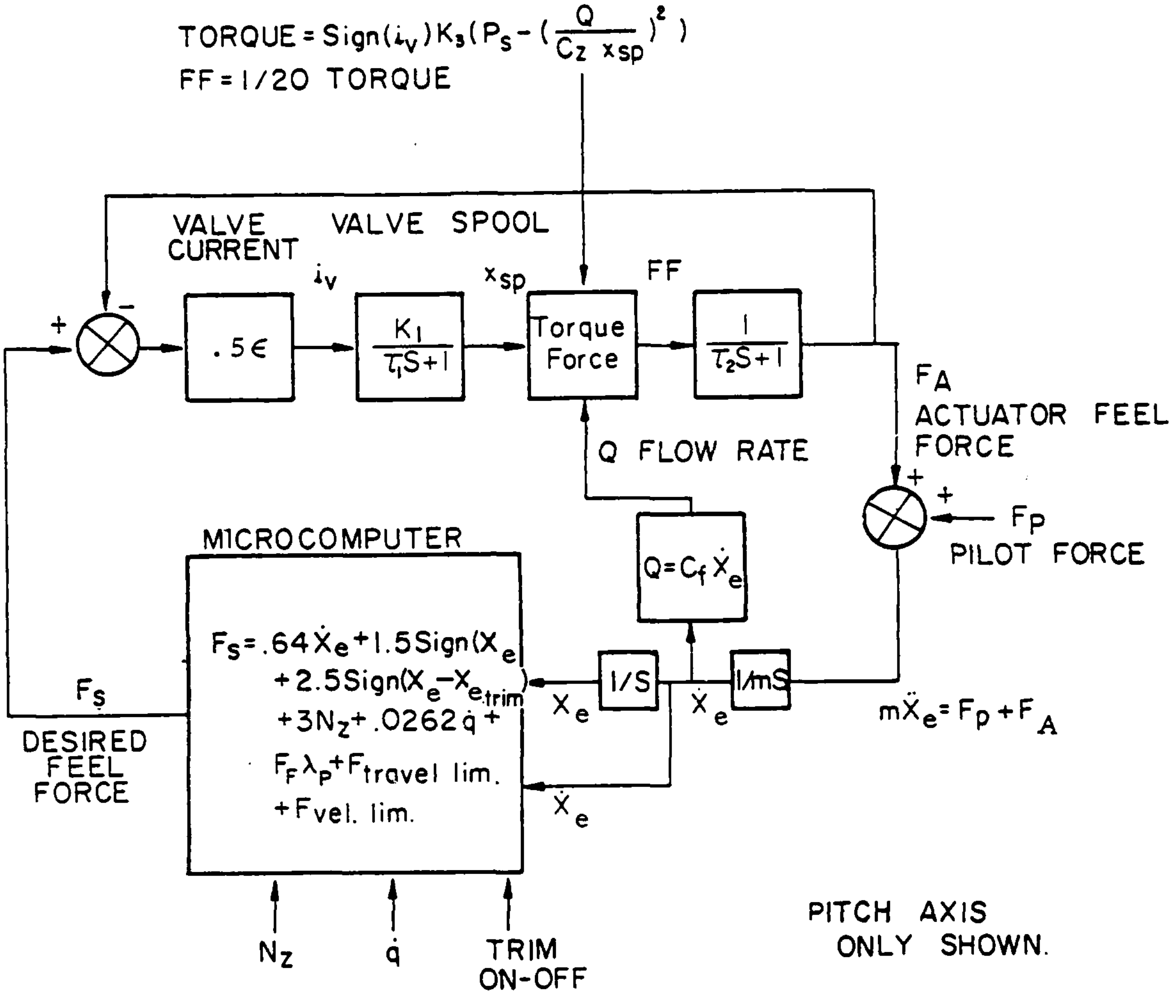


Figure 4.4 Control Loading Loop

SECTION 5
COMPUTER PROGRAM SYSTEM DESCRIPTIONS

The entire control loader software was written in the Intel high-order microcomputer language called PLM (PL/M-80), with the exception of the analog-to-digital conversion routine which was performed in assembly language. The programs were structured into a utility module, a scan module, and a simulator module. As the name implies the utility module provided system support in the form of calibration and test routines, system interface procedures, and system initialization. The scan module causes twelve of the available sixteen A-to-D converter channels to be scanned sequentially and the converted values to be placed in the appropriate RAM locations for later use. The simulator module contains the frame rate generator, simulator initialization routines, and the simulator equations themselves.

5.1 GENERAL PROCESSING

Processing starts in the utility module after a reset (push button reset or power-on reset). As illustrated in Figure 5.1 the system hardware is initialized first. The simulator module is entered next which, in turn, initializes the needed equation parameters before starting through the simulator loop. If no request is pending (at the teletype keyboard) the loop is executed 120 times per second, solving equations for all three axes. If a request for a change is made by the system operator, he has the option to exit the simulator processing and enter the executive. If he elects to make a parameter change he can continue to make as many changes as he desires before re-entering the simulator loop. If the operator chooses to leave the simulator, control will pass to the executive which will request a program number. Programs 0 through 4 are self test and calibration routines which, once completed, will pass

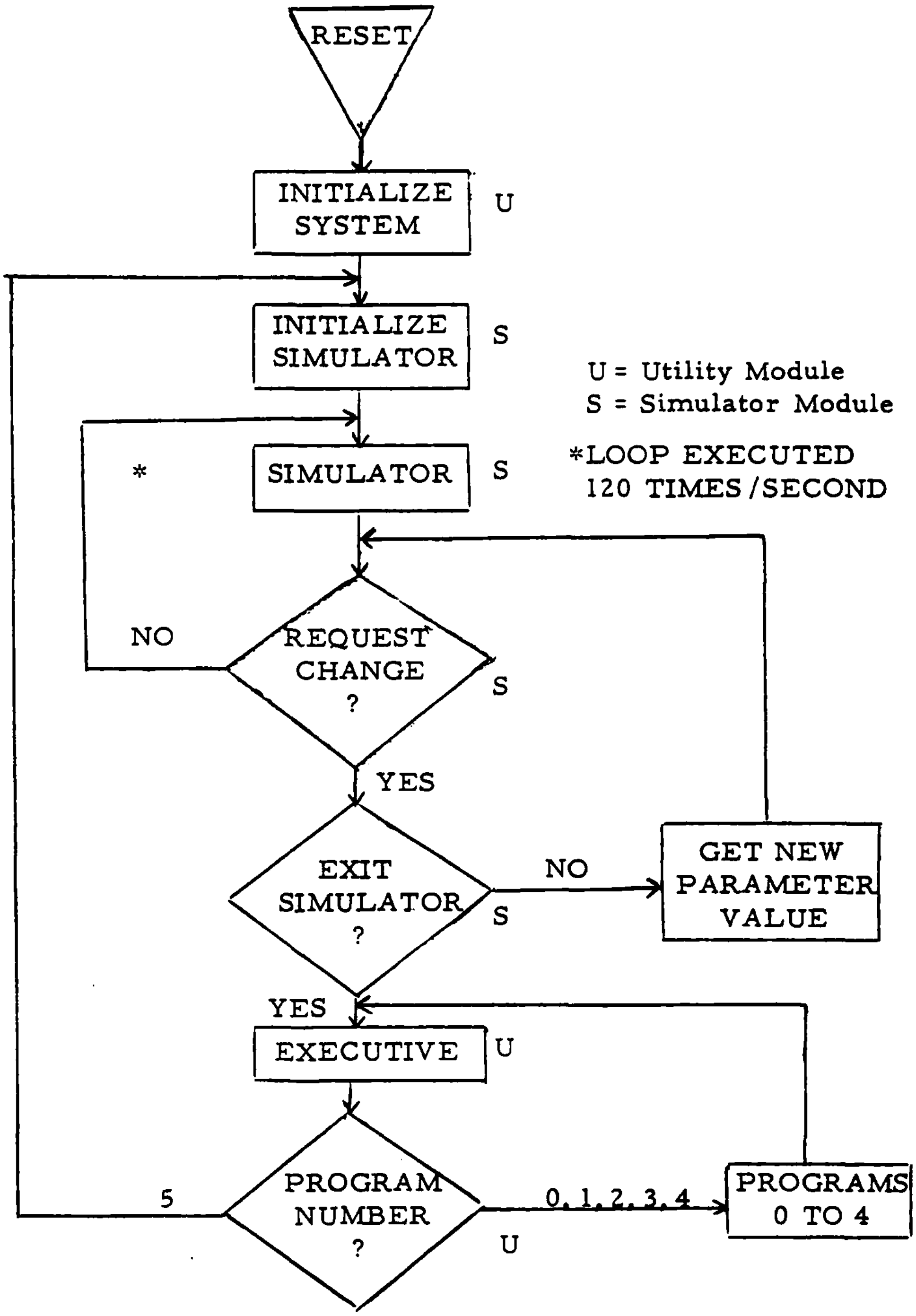


Figure 5. 1 General Flowchart

control back to the executive. Program 5 is the simulator and, if selected, will re-initialize all the simulator parameters to their original value before entering the simulator loop.

5.2 THE UTILITY MODULE

The utility module contains the main program for the system. This program initializes the appropriate hardware and software items in a procedure called INITIALIZE. It then enters (calls) the simulation loop contained in the Simulator Module. However, if the operator terminates the simulation, the system then returns to the Utility Module and requests a command from the console operator by typing:

```
CONTROL LOADER EXECUTIVE
WHICH PROGRAM?
```

Depending on the console operator's input, it calls the appropriate procedure. These procedures include various test and calibration routines as well as the simulator procedure itself. The table below lists the procedure which will be executed when a particular input is supplied

INPUT	PROCEDURE
0	MATHBOARD TEST
1	ADC TEST
2	GAIN TEST
3	DAC TEST
4	RAMP TEST
5	SIMULATOR

The following subsections discuss the various elements and procedures in the Utility Module. Flowcharts for these Utility Module procedures are contained in Appendix A. The above mentioned procedure INITIALIZE performs initialization functions for many different hardware items. It is therefore more efficient to discuss the portions of INITIALIZE with those respective items.

5.2.1 Declarations

The utility module contains two kinds of declaration statements "literal" and "type." The literal definitions can be observed in the program listing and are merely macro or literal word substitutions for certain quantities. They are grouped according to function and they also appear in the simulator module so that the same words may be used in both modules. For example, the teletype input port is defined by setting the word CONSOLE$IN functionally equivalent to the SBC 80/20 port number of 'EC.'

The type declarations are standard PLM statements which declare the variable names as being either "byte" or "address" types. All of the variables are declared as being public so that the simulator module may share the locations.

5.2.2 Teletype Communications

During the initialization subroutine, INITIALIZE, the various functions are performed to set up the communication section of the hardware for teletype usage. There are two hardware components involved in this section: the 8251 Programmable Communication Interface (PCI or USART) and the 8253 Programmable Interval Timer (PIT).

Only one of the counters in the 8253 (counter #2) is used for communication purposes; it generates the baud rate for the PCI. Counter #2 is dedicated to the baud rate function and is initialized by first writing the appropriate control word to the 8253 followed by the "divide-by" count (see Section 3.9.1 and 3.9.2 of the System 80/20 Hardware Reference Manual). The control word was synthesized to reflect the following attributes:

Select Counter #2	(10)
Load LSB, MSB	(11)
Mode 3	(11)
Binary Counter	(0)

Therefore, the control word B6 (hex) is output to the control word register at port DF. The baud rate factor ("divide-by" count) is next written to the counter register (port DE), low byte first. This baud rate factor is 0263_{16} or 611_{10}. It is determined by taking the closest integer value which results from a calculation involving the desired baud rate (110 bits/sec), the 8251 baud rate multiplier (16) and the basic clock period (930 ns). Thus the baud rate factor is

$$\mathrm{BRF} = \frac{1}{110 \times 16 \times 930\mathrm{ns}} = 610.95 \text{ or } 611$$

With the BRF = 611 the actual baud rate is 109.99 bits per second - very close to 110.

Similarly the 8251 USART is initialized with two words - a mode word and a control word (see Section 3.7, 80/20 Hardware Reference Manual). The mode word (CE hex) was synthesized for the following functions:

Baud Rate Multiplier = 16	(10)
Character Length = 8 bits	(11)
Parity Disabled	(0)
Parity Odd	(0)
Stop bits = 2	(11)

The control word is 35_{16} and was synthesized for the following functions:

Transmit Enable	(1)
Data Terminal Not Ready	(0)
Receive Enable	(1)
Send Break-Normal	(0)
Error Reset	(1)
Request to Send Active	(1)
Internal Reset	(0)
Hunt Mode Disabled	(0)

Both the mode word and the control word are written to port ED; the mode word must be written first. As described in the 80/20 references, the data port is EC.

5.2.3 Console Input/Output

The communications which are conducted between the system and the operator's teletype (TTY) is provided by the following procedures:

CONSOLE$OUTPUT
CONSOLE$INPUT
HEX$OUT
HEX2$OUT
LINE$INPUT
PRINT
CRLF
FIX$TO$ASCII
ASCII$TO$FIX

The desired communication proceeds one character at a time and thusly the CONSOLE$INPUT and CONSOLE$OUTPUT procedures perform their respective functions on one character. If the character to be output is a hexadecimal value (4-bits) the procedure HEXOUT is used to convert the value to an ASCII character and subsequently call CONSOLE$OUTPUT. Similarly, in HEX2$OUT, a single byte (8-bits) is converted to two ASCII characters which are output in succession, high order first. If an entire line of input is required from the operator, the procedure LINE$INPUT is executed which places the ASCII character images sequentially into an array called LINE$BUF and also provides the calling routine with the character count. The line is terminated by the operator with a carriage return. On the other hand, if a line of text is to be displayed on the console, the PRINT procedure is used. This procedure requires the address of the first character to be displayed. All characters are sequentially output to the console until an "ETX" (03) is encountered. The CRLF procedure outputs a carriage return (CR) and a line feed (LF) to the console when it is invoked.

The FIX$TO$ASCII and ASCII$TO$FIX procedures communicate with the console operator and convert data between a string of ASCII numeric characters and the equivalent 16-bit value. FIX$TO$ASCII takes the value of a parameter called RESULT, converts it to a string of ASCII numeric characters and outputs the string to the console (preceeded by a message such as "RESULT ="). ASCII$TO$FIX performs the opposite function. It accepts a string of decimal characters from the console (followed by a CR), converts the string to a hexadecimal value and sets RESULT equal to that value. These two procedures make extensive use of the above mentioned procedures as well as the EXECUTE procedure which is discussed later in this section. The method employed is to successively divide (or multiply) the parameter by a factor of 10. For example, in FIX$TO$ASCII the hexadecimal value to be converted is divided by "multiplier" of 10000 (since the maximum hexadecimal value is 65,536). The result of the division is a decimal integer (0 to 9) which forms the first output character. That character (times the multiplier) is then subtracted from the original value leaving the lower order digits. The "multiplier" is then reduced by a factor of ten down to 1000. This routine of divide-subtract-divide is repeated four more times to provide 5 digits in all. The five digits are then output to the console. ASCII$TO$FIX works in just the opposite order: it accepts up to five digits, performs a multiply-add-multiply routine and places the value in the parameter RESULT.

5.2.4 Mathboard Interface

The SBC 310 High Speed Mathboard was supplied by the manufacturer with an I/O base address of C8 selected. (See SBC 310 HRM, paragraphs 2-8, 2-9.) This address was used without change and the literal declarations reflect this base address. The memory base address assignment is under software control and was established as address FFFO hex at the logical top of memory. (See SBC 310 HRM, chapter 3.) The various mathboard functions (e.g. fixed point multiply) were also defined in literal declaration (e.g. MUL = 0) for later use. (See SBC 310 HRM, Table 3-2.) The two argument registers of the math board are 32 bits in length which requires two address-type parameters for data transfer. The "operand" register (which is also the "result" register) is defined by two names: OPERAND$IL and OPERAND$IH for the low and high halves respectively. Similarly, the "operator" register is defined by the names OPERAND$2L and OPERAND$2H. All four of these parameters are "based" address-type variables with the addresses assigned in the INITIALIZE procedure. (See SBC 310 HRM, chapter 3.)

The interface between the mathboard and the system is handled by three procedures:

- PERFORM
- EQUATE
- EXECUTE

PERFORM causes the mathboard to take the action which is defined by the parameter OPERATION. For example if OPERATION = 0 a fixed point multiply will be performed on the values which are in the argument registers. It then waits for the operation to be complete and checks the error flags. (See SBC 310 HRM, paragraph 3-6.) If no error resulted, it returns to the calling routine. If an error was detected by the mathboard (e.g. division by zero), PERFORM causes an error message to be displayed on the console which states the status code, opcode and the value of the two operands.

EQUATE is a procedure which handles floating point operations (32-bit operands) whereas EXECUTE handles fixed point operations (16-bit operands). Both of these routines proved to be too expensive to use in real-time but they are employed in the MATHBOARD$ TEST and ASCII conversion procedures. EQUATE requires three address pointers to the 32-bit (4-byte) locations of the destination, operand and operator for the floating point function. It also requires the operation to be defined by setting OPERATION to the appropriate value. EQUATE uses PERFORM once the operand and operator values are loaded in the argument registers. The result is placed in the destination location. For example

```
CALL EQUATE (.RESULT, .RESULT, FPLUS, .TEMP)
```

will produce the same effect as the following FORTRAN statement:

```
RESULT = RESULT + TEMP
```

EXECUTE operates in a similar manner except that the actual 16-bit values are passed as opposed to addresses. Thus,

```
RESULT = EXECUTE (RESULT, TIMES, TEMP)
```

has the same effect as the following FORTRAN statement:

```
RESULT = RESULT * TEMP
```

5.2.5 Analog Input Interface

The analog input chores are performed by the SBC 711 Analog Input board and two software subroutines: ADVAL in the Utility Module and SCAN in the Scan Module. ADVAL returns binary value of the channel designated in its argument list. SCAN is a routine which permits a range of channels to be converted in a sequential manner.

The SBC 711 board had to be modified by employing the user selectable options provided. This board, therefore, has the following characteristics, some of which were modified from the OEM configuration. The base address remains F700, as supplied. In addition, the full scale range of ± 10v was not changed. The ADC trigger options were not used because the trigger is under software control. The transfer acknowledge delay was not changed from the preset 50 ns. However, the "offset binary code" option was selected by reconfiguring the jumpers around pin 67. (see SBC 711 HRM, paragraph 2-10).

The software is structured around "based" variables AICOM, AIFCR, AILCR, AIINT and AIVAL referring to the command register, first channel register, last channel register, interrupt register and data registers of the SBC 711, respectively. (See Section 3 of SBC 711 HRM.) The command register is initially reset in subroutine INITIALIZE.

Subroutine ADVAL merely follows the recommended procedures for a random channel conversion; that is, AIFCR is loaded with the desired channel, the conversion is started through AICOM, the subroutine waits for the status register (AICOM) to indicate EOC, the conversion is stopped, and the value returned. ADVAL is used in the ADCTEST (program #1) subroutine which is the PLM equivalent of the calibration routines ADCRNG and ADCOFF in the SBC 711 HRM (Section 5-5, Calibration Procedures).

For the simulator function, a sequential scan of the desired channels is performed. A full description of this process is contained in the Scan Module section.

5.2.6 Analog Output Interface

The analog output chores are handled by Intel's SBC 710 Analog Output Board. The analog output registers on this board appear as simple contiguous memory locations to the CPU. The base address F708 hex was factory prewired and not changed. The process of outputing an analog signal is one of merely writing the desired hexadeciaml value to the appropriate location. The variables DAC0, DAC1, DAC2 and DAC3 are based address-type parameters which refer to the analog output registers. Thus, if the value of RESULT is to be converted to an analog voltage on channel 0, the PLM statement is merely

```
DAC0 = RESULT
```

The only physical modification to the SBC 710 was to select the "offset binary code" option as described in Section 3.1.3.

5.2.7 Self Test and Calibration Routines

The Utility Module contains five procedures which were written to provide a means of testing all three of the peripheral circuit boards (SBC 711, SBC 724, SBC 310) and of calibrating the analog input and analog output boards (SBC 711 and SBC 724 respectively). The analog reference manuals contain specific information pertaining to the calibration of the various amplifiers and since these units are software driven the manuals also contain sample programs which can be used for the calibration procedure. The procedures contained in the Utility Module are PLM equivalents of the ones in the manuals. The system operator selects the desired test program on the console by responding to the Utility Module's request which appears after leaving the simulator loop:

```
CONTROL LOADER EXECUTIVE
WHICH PROGRAM?
```

The operator responds by typing in an integer from 0 to 5 which has the following meaning:

Program	Name
0	Mathboard Test
1	ADC Test
2	Gain Test
3	DAC Test
4	Ramp Test
5	Simulator

The simulator function will be discussed in Section 3.2.5. The other functions are described in the following paragraphs.

5.2.7.1 Mathboard Test (0)

The hardware reference manual for the high-speed mathboard (SBC 310) does not provide any means of testing this unit. Of course, there is no calibration required since it is entirely digital. Therefore, a procedure MATHBOARD$TEST was written which will accept an integer from the console in the range of 0 to 65535 (2^{16}).

This integer is used to check each mathematical function which the mathboard is capable of performing which includes integer functions, floating point functions, and the conversions between these functions. That is, the value supplied by the operator is multiplied by itself, divided by itself, squared, square-rooted, etc. At the end of the sequence of tests, the last value is printed on the console and will correspond to the value supplied by the operator if the test is passed. If the value typed in by the operator is 0 (zero), the system will type two error messages since division by zero was performed twice, once in integer and once in floating point. These error messages relate the following:

a) Status (error) code - See SBC 310 HRM, p 3-5

b) Opcode which caused error

c) The contents of the two argument registers after the error was detected

The operator may also request that the above status information be printed each time the mathematics board performs an operation during the test. If this is the case, a variable called ERR$SW is set equal to 1; otherwise, ERR$SW = 0. The procedure PERFORM recognizes this variable and acts accordingly in typing out the status information. Refer to Section 3.3 for a complete description of the console interaction.

This procedure makes use of the EQUATE, EXECUTE and PERFORM procedures described previously. It also uses the various console input/output routines and also employs the ASCII conversion routines ASCII$TO$FIX and FIX$TO$ASCII, all of which were previously discussed. Thus the MATHBOARD$TEST procedure is structured almost exclusively from pre-defined routines.

5.2.7.2 ADC Test (1)

The analog input board hardware reference manual provides calibration programs which support the required calibration procedures (see SBC 711 HRM, chapter 5). However, these programs (SBC 711 HRM, Table 3-3) are written in assembly language. Therefore the PLM procedure ADCTEST was written which incorporates all of the features of the assembly language program ADCOFF (ADCRNG is another name for the same program). This program is used to perform the calibrations associated with offset and range adjustments (SBC 711 HRM, paragraphs 5-7 and 5-8 respectively).

Specifically, this procedure asks the operator which analog input channel he wishes to look at (channels 0 through F hex). The procedure then continuously reads and displays (in hex) the digital value of the analog input until a character (any character) is typed on the console. Control then passes back to the executive. The net effect which this procedure produces is that of a digital voltmeter which types the values out in hexadecimal.

5.2.7.3 Gain Test (2)

The GAINTEST procedure was written to satisfy the need of performing a gain adjustment on the programable amplifier of the analog input board (SBC 711 HRM, paragraph 5-6). This procedure incorporates the same features as the assembly language program PGAADJ described in the hardware reference manual. When selected this procedure merely types "PGA GAINTEST" on the console and then enters a gain switching loop until a character (any character) is typed on the console.

5.2.7.4 DAC Test (3)

The analog output board is similar to the analog input board in that its amplifiers must be calibrated in accordance with published procedures (SBC 724 HRM, paragraph 5-6). The PLM procedure DACTEST was written to fulfill the software requirements for such a calibration. When selected, it types the message

DACTEST
HEX VALUE?

to which the operator must respond with four hex characters such as 5B2C. This hex value is then written out to all four DAC channels which in turn convert the value to analog voltages at the output. Thus, if the operator wishes to have a positive full scale voltage (+10v) at the output he must type 7FF0. If he wishes to have a negative full scale voltage (-10v), he must type FFF0. The hex values reflect the offset binary (2's complement) format selected under the hardware modifications. The last digit is zero because only the most significant twelve bits placed in the DAC are converted (12-bit accuracy). Control passes back to the executive immediately after execution.

5.2.7.5 Ramp Test (4)

The procedure RAMPTEST was written to provide a dynamic signal on the DAC output channels. This signal is a sawtooth ramp with a period of approximately five seconds. It is achieved by simply incrementing a variable and writing its value out to all four DAC's in a continuous loop.

5.2.8 Interupt Structure

During the initialization subroutine INITIALIZE three control words are written to the 8259 Programmable Interrupt Controller (PIC). The first word is initialization control word ICW1-C which sets the format interval to 8 (bit F = 0) and the single/slave flip-flop to single (bit S = 1). (See p 3-107, 80/20 HRM.) This word also transfers three address bits (A_5, A_6, A_7) to the PIC. The second word is ICW2 which transfers the remaining address bits (A_8 - A_{15}) to the PIC. The address bits represent the location of the interrupt jump table in the software; the PIC inserts the lowest address bits (A_0 - A_4) automatically based on the interrupt being services. At this writing, the PL/M compiler requires interrupt routines at the standard locations, despite the capabilities of the 8259. The third word written is the operation control word OCW1 *which merely enables the desired interrupts.* These words are written to the ports which are designated by the hardware configuration of the 80/20; that is, ICW1-C is written to port 0DAH, ICW2 and OCW1 to port 0DBH.

At the end of each interrupt service routine the "non-specific end-of-interrupt" control word OCW2-E (20H) is written to port 0DAH. This action clears the particular interrupt in-service (IS) bit in the 8259 corresponding to the assigned interrupt level.

For a full discussion of the 8259 PIC and to ascertain the implications of the above actions refer to Section 3.10.1 (pages 3-87 to 3-110) of the SBC 80/20 Hardware Reference Manual. Of particular interest is the information on pages 3-109 and 3-110 which describes how the interrupts can be configured on the 80/20 board itself. The interrupt structure is used only for frame rate generation which is discussed in Section 3.2.5.2.

5.3 SCAN MODULE

The Scan Module is an assembly language program which is invoked by the simulator module when the values of the analog input channels are to be read. The conversion of the 12 analog input channels was previously accomplished with an interrupt-driven PLM procedure (subroutine) and consumed approximately 3 MS of processing time. No advantage was realized from the interrupt structure since the convert time (37 μS per channel) was much shorter than the data handling software time. Additionally, the machine code resulting from the PLM statements was rather inefficient. Therefore, the input scan routine was converted to a non-interrupt, assemply language program which is capable of converting all 12 analog channels in less than 1 millisecond.

A declare statement in the simulator module establishes the following address-type (16-bit) variables in contiguous locations in RAM: PDOT, PITCH, NZ, QDOT, ROOT, ROLL, NY, PPDOT, YDOT, YAW, PTRIMSIG, RTRIMSIG. The order in which they are declared is the order in which they occur in memory. The scan module causes input channels 0 through 11 to be converted consecutively with the data from channel 0 being placed in PDOT and the subsequent data values in the scan to be placed in consecutive memory locations (2 bytes each). As a result the above mentioned variables take on the value of the corresponding analog input channel.

The scan process is conducted in its entirety when a "call" is made from the simulator module. The simulator module first initializes the first-channel-register and the last-channel-register. It enables the auto increment feature and then starts the first conversion. (See Section 3.2.3.5 for a discussion of the registers involved.) The "call" is then made to SCAN. The first scan module operation is to initialize the 8080's DE register with the address of PDOT and the B register with the channel counter (12). The scan loop is then entered which starts by checking the analog input board to see if a conversion is in process, in which case the processor will wait in a loop until the conversion is complete. The channel value is read (placed in the HL register) and a new conversion is started by writing a 03 hex out to location F700, the analog input base address: The value in the HL register is then written out to the destination specified by the DE register which is incremented in the process. Register B, the channel counter, is decremented to end the loop and transfer control back to the status check. When register B reaches zero, the converter is reset and control is transferred back to the simulator module.

5.4 SIMULATOR MODULE

The simulator module performs all the tasks necessary to run the real-time simulation. It contains its own declarations (in addition to those of the utility module), frame generation software, dummy procedures, lead and lag procedures for each axis, initialization statements and the simulation loop itself. The simulator is entered immediately after a RESET and is also one of the programs (program 5) which the operator can select under executive control. Upon entering the simulator, all variable parameters are initialized and the simulator loop is then entered. The loop is executed 120 times each second, calculating the forces for all three axes in each loop. The operator may request a parameter change in which case the simulation is halted until the desired changes are made. The operator may also enter the executive program at this point. The following paragraphs describe the procedures associated with the Simulator Module. The corresponding flowcharts are contained in Appendix A.

5.4.1 Declarations

Most of the declarations in the simulator module are identical to those of the utility module and actually reference them by the EXTERNAL attribute. Additional parameters are declared which are unique to the simulator module. The order in which these variables are declared governs the order in which they reside in memory and, therefore, the order in which they are referenced. For instance, the order of the 11 parameters following PDOT in the declaration determines the "parameter number" for each when a change is requested.

5.4.2 Frame Rate Generation

The frame timing is generated by loading counter #0 of the 8253 with the appropriate "frame rate factor" (FRF), counting down to zero, and allowing the 8253 to interrupt the processor at the terminal count (zero). The processor, therefore, is free to perform simulator tasks while the 8253 is counting down. The clock for this function is proportional to the crystal controlled clock for the microprocessor.

That is, the micro's clock is divided by 2 before use by the counter, resulting in a counter clock period of 930 nanoseconds. For a frame rate of 120 frames/seconds the frame rate factor is

$$\mathrm{FRF} = \frac{1}{120 \text{ X } 930 \text{ ns}} = 8960.57$$

The counter, therefore, is loaded with a value of 2301_{16} (8961_{10}).

Prior to operation, the 8253 is loaded with a command word in the INITIALIZE procedure. This command word is one which was synthesized in order to program counter #0 of the 8253. This word has a value of 30 hex which was produced from the following attributes:

Select Counter #0	(00)
Load LSB, MSB	(11)
- - -	(0)
Mode 0	(00)
Binary Count	(0)

(See Section 3.9 of the 80/20 HRM for details of the 8253 operation.)

The frame rate function has been assigned the highest available interrupt priority level - level 1. Therefore, several items had to be considered to accommodate this interrupt. First, the output of 8253 counter #0 was connected to interrupt request 1 (IR1) of the 8259 Programmable Interrupt Controller. This was accomplished by connecting together jumper pins 25 and 35 on the 80/20 CPU Board. Second, the interrupt mask bit (bit 1) of the 8259 must be unmasked (set to 0) by writing the appropriate word in the proper sequence to port DB. (See Section 3.10 of the 80/20 HRM.) Thirdly, an interrupt service routine had to be written to handle the interrupt once it occurs.

The interrupt service routine "FRAME" uses the PLM interrupt procedure declaration so that the first interrupt instruction is in memory location 0008. This routine provides the following functions:

1. Restarts the frame counter by writing the frame rate factor FRF to port DC, low byte first.

2. Checks for framing errors. The frame rate switch FRSW is set to zero at the end of the simulator block. If FRSW = 1 when interrupt 1 occurred, simulator processing was not complete and the on-board LED is flashed to indicate the framing error.

3. The frame rate switch FRSW is set to 1.

4. The logic level bit 0 at J1 (port E4) is inverted to indicate frame time available.

5. The non-specific end-of-interrupt word (20 hex) is written to port DA to reset the 8259 "in-service" byte 1 (IS1).

Framing is accomplished through the use of the parameter FRSW. At the beginning of each frame (first statement after the label SIMLOOP in the listing) FRSW is set to zero. At the completion of the equation processing the processor waits in a loop for FRSW to change from zero. This occurs in the interrupt routine FRAME so that when control returns to the "idle loop" FRSW = 1 and the loop is exitted.

A unique feature about this control loading simulator is the service provided by bits 0 and 1 at the digital interface on the SBC 80/20. Bit 0 is inverted each time FRAME is executed (120 times per second) and, when connected to an oscilloscope, it indicates the amount of frame time available. Bit 1 is set high at the beginning of each frame (near the label SIMLOOP in the listing) and it is set low at the completion of the equation processing and thus it indicates the amount of frame time consumed.

5.4.3 Dummy Procedures

The simulator module employs several of the procedures which were defined in the utility module, namely PRINT, PERFORM, ASCII$TO$FIX and FIX$TO$ASCII. Furthermore the SCAN procedure from the scan module is also invoked. The dummy procedures declare that the actual coding resides "external" to the simulator module. (See Section 8.1, PL/M-80 Programming Manual.)

5.4.4 Simulator Processing

When a RESET occurs or when program 5 is selected by the operator in executive mode the procedure SIMULATOR is called. The simulator variables and coefficients are first initialized to the appropriate values, and the frame generator is started. At this point the processor enters the simulator loop, SIMLOOP. The first action taken is to perform a complete input of the variable parameters through the routine SCAN. Once this task is accomplished the processor begins to calculate the output forces: pitch, roll and yaw in that order. The processing for each axis proceeds in a manner as shown in Figure 5.2. As each force is calculated, its value is written out to the respective DAC and processing moves on to the next axis. After the yaw computations are complete and the yaw force has been written out to the yaw output channel, a loop is entered which causes the simulator to wait for the end-of-frame interrupt as explained in Section 5.4.2. The simulator then checks for a request from the system console (teletype). If no request is pending, control passes back to the beginning of the simulator loop. If a request is pending (a key, any key, was depressed), the operator is asked whether a parameter change is being requested. If the reply is negative, control passes back to the executive in the utility module. If the reply is affirmative a parameter change routine is entered which allows the operator to examine the present value of and change the value of as many parameters as is necessary. Upon leaving this routine control passes back to the beginning of the simulator loop. Each of these blocks will be discussed in detail within the following paragraphs.

5.4.4.1 Simulator Initialization

When control passes to the simulator as a consequence of a reset or the selection of program 5 in the executive all of the necessary simulator parameters are initialized to their original value. These parameters all have storage locations in ramdom access memory

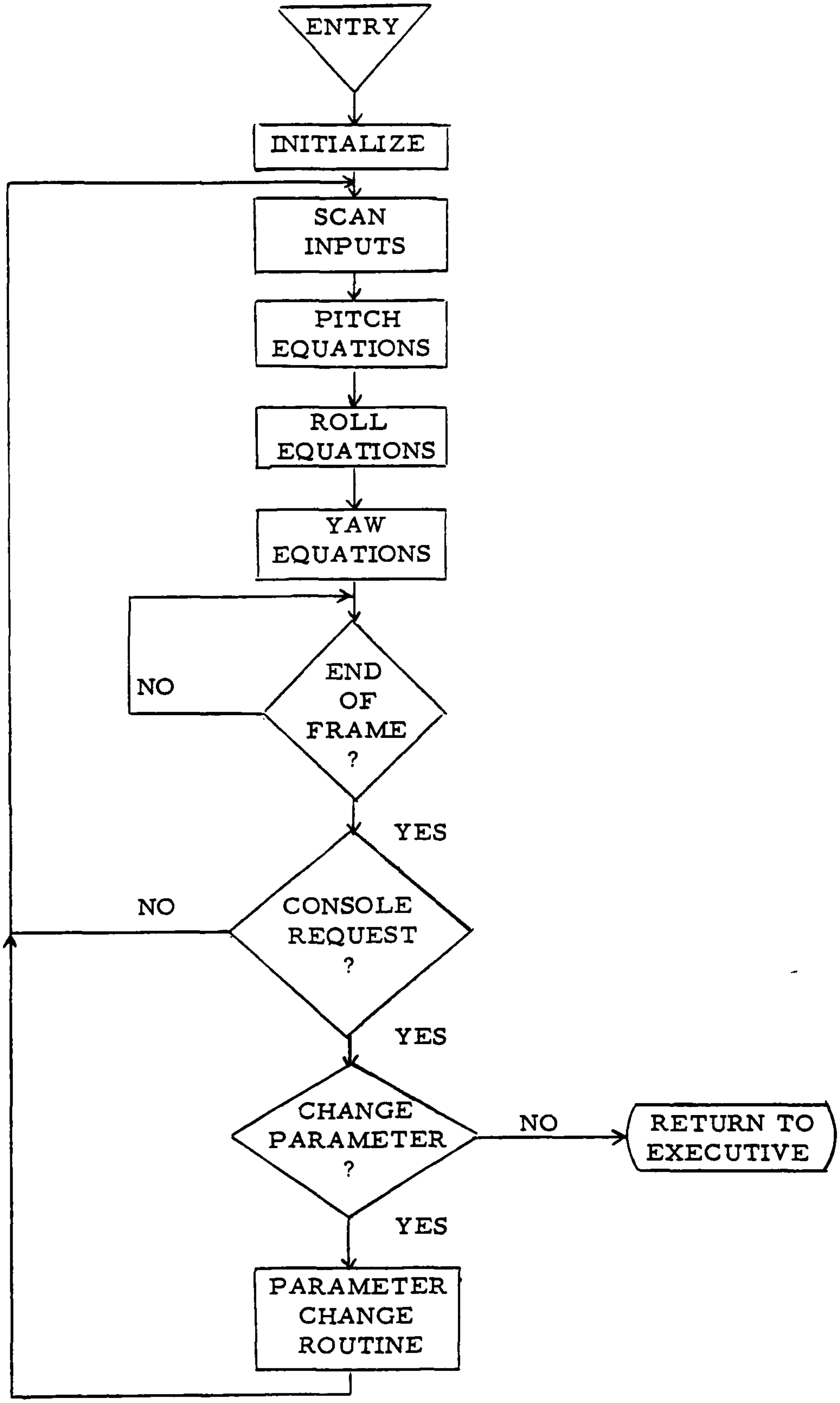


Figure 5.2 Simulator Module Flowchart

(RAM) so that they may be changed by the operator as necessary. Furthermore, all of the changeable coefficients reside in contiguous RAM locations so that they may be selected by number for a change. (See Section 6.6.3.2.) The "by-the-number" method was chosen over a "by-name" method due to its very simple implementation scheme and very significant reduction in storage requirements. The order in which they appear in the declaration statements determines the respective reference numbers. There are a total of 224 such parameters but not all are used in the present simulation because the full capability of setting ten spring breakpoints is not fully utilized. These parameters and their respective numbers are included in Table 5.1 for completeness.

Several other parameters are also initialized which are used during processing for intermediate and final value storage. Furthermore the frame generation is started by a simple call to the FRAME procedure. This procedure, as described in Section 5.4.2, loads the hardware counter with the frame rate factor which starts the frame timing. When the counter reaches zero an interrupt occurs which causes this FRAME procedure to be executed again but this will only occur at the end of the simulator loop. The initialization process is concluded when all of the bits on output port E4 (hex) are set low. Bits 0 and 1 are used to monitor the frame times as described in Section 3.3.

5.4.4.2 Simulator Loop

When control passes to the beginning of the simulator loop at SIMLOOP, a small amount of processing is performed in preparation for performing the scan of the analog input channels. The first operation is to set the frame rate switch FRSW = 0. This allows the simulator to pause at the conclusion of the calculations until the FRAME interrupt occurs to set FRSW = 1. The next operation sets the "frame time used" bit, Bit 1 at J1, to a high level. This bit will be set low at the conclusion of the simulator calculations as an indication of frame time used. The next few statements initialize the analog input

```
            /* THE FOLLOWING COEFFICIENTS ARE IN ORDER FOR CHANGE IF NECESSARY
               THE NUMBER AFTER EACH IS THE NUMBER USED FOR CHANGE. */
 2               KPVISC = 13979;            /* (0)      PITCH VISCOUS DAMPING COEFFICIENT (.2173) */
 2               KPCOUL = 328;              /* (1)      PITCH COULOMB FRICTION COEFFICIENT (1.5) */
 2               KPBRAK = 546;              /* (2)      PITCH BREAKOUT FORCE COEFFICIENT (2.5) */
 2               KNZ = 13107;               /* (3)      PITCH NORMAL ACCELERATION COEFFICIENT (.2000) */
 2               KQDOT = 458;               /* (4)      PITCH ACCELERATION COEFFICIENT (.0035) */
 2               PDBANDA = 0;               /* (5)      PITCH DEADBAND AFT */
 2               PDBANDF = -PDBANDA-1;      /* (6)      PITCH DEADBAND FORWARD */
 2               PLIMFOR = -8847;           /* (7)      PITCH LIMIT FORWARD (2.7 IN) */
 2               PLIMAFT = 18678;           /* (8)      PITCH LIMIT AFT (5.7 IN) */
 2               KPSNA = 3;                 /* (9)      NO. OF AFT PITCH SPRING COEFFICIENTS */
 2               KPSNF = 2;                 /* (10)     NO. OF FORWARD PITCH SPRING COEFFICIENTS */
 2               KRSNL = 1;                 /* (11)     NO. OF LEFT ROLL SPRING COEFFICIENTS */
 2               KRSNR = 1;                 /* (12)     NO. OF RIGHT ROLL SPRING COEFFICIENTS */
 2               KYSNL = 1;                 /* (13)     NO. OF LEFT YAW SPRING COEFFICIENTS */
 2               KYSNR = 1;                 /* (14)     NO. OF RIGHT YAW SPRING COEFFICIENTS */
 2               KPSOFFA(0) = 0;            /* (15)     AFT OFFSET 1 */
 2               KPSOFFA(1) = 851;          /* (16)     AFT OFFSET 2 */
 2               KPSOFFA(2) = 1373;         /* (17)     AFT OFFSET 3 */
 2               KPSBRKA(0) = 0;            /* (25)     AFT BREAKPOINT 1 */
 2               KPSBRKA(1) = 4260;         /* (26)     AFT BREAKPOINT 2 */
 2               KPSBRKA(2) = 11469;        /* (27)     AFT BREAKPOINT 3 */
 2               KPSPRINGA(0) = 23522;      /* (35)     AFT SPRING COEFFICIENT 1 */
 2               KPSPRINGA(1) = 10427;      /* (36)     AFT SPRING COEFFICIENT 2 */
 2               KPSPRINGA(2) = 7445;       /* (37)     AFT SPRING COEFFICIENT 3 */
 2               KPSOFFF(0) = 0;            /* (45)     FORWARD OFFSET 1 */
 2               KPSOFFF(1) = 341;          /* (46)     FORWARD OFFSET 2 */
 2               KPSBRKF(0) = 0;            /* (55)     FORWARD BREAKPOINT 1 */
 2               KPSBRKF(1) = 4915;         /* (56)     FORWARD BREAKPOINT 2 */
 2               KPSPRINGF(0) = 21840;      /* (65)     FORWARD SPRING COEFFICIENT 1 */
 2               KPSPRINGF(1) = 17300;      /* (66)     FORWARD SPRING COEFFICIENT 2 */
 2               KRVISC = 2359;             /* (75)     ROLL VISCOUS DAMPING COEFFICIENT */
 2               KRCOUL = 492;              /* (76)     ROLL COULOMB FRICTION COEFFICIENT (1.5) */
 2               KRBRAK = 655;              /* (77)     ROLL BREAKOUT FORCE COEFFICIENT (2.5) */
 2               KNY = 32768;               /* (78)     ROLL NORMAL ACCELERATION COEFFICIENT (.2000) */
 2               KPPDOT = 210;              /* (79)     ROLL ACCELERATION COEFFICIENT (.0035) */
 2               RDBANDR = 0;               /* (80)     ROLL DEADBAND RIGHT */
 2               RDBANDL = -RDBANDR-1;      /* (81)     ROLL DEADBAND LEFT */
 2               RLIMRT = 10158;            /* (82)     RIGHT ROLL LIMIT (3.1 IN) */
 2               RLIMLT = -10158;           /* (83)     LEFT ROLL LIMIT (3.1 IN) */
 2               KRSOFFR(0) = 0;            /* (84)     RIGHT ROLL OFFSET 1 */
 2               KRSBRKR(0) = 0;            /* (94)     RIGHT ROLL BREAKPOINT 1 */
 2               KRSPRINGR(0) = 29491;      /* (104)    RIGHT ROLL SPRING COEFFICIENT 1 */
 2               KRSOFFL(0) = 0;            /* (114)    LEFT ROLL OFFSET 1 */
 2               KRSBRKL(0) = 0;            /* (124)    LEFT ROLL BREAKPOINT 1 */
 2               KRSPRINGL(0) = 29491;      /* (134)    LEFT ROLL SPRING COEFFICIENT 1 */
 2               KYVISC = 14828;            /* (144)    YAW VISCOUS DAMPING COEFFICIENT [illegible] */
 2               KYCOUL = 696;              /* (145)    YAW COULOMB FRICTION COEFFICIENT (4.25) */
 2               KYBRAK = 819;              /* (146)    YAW BREAKOUT FORCE COEFFICIENT (5.0) */
 2               YDBANDR = 0;               /* (147)    YAW DEADBAND RIGHT */
```

TABLE 5.1 PARAMETER NUMBER TABLE

```
PL/M-80 COMPILER    CONTROL LOADING PROGRAM - SIMULATOR MODULE (CNTS85)

 191   2          YDBANDL = -YDBANDR-1;        /* (148)    YAW DEADBAND LEFT */
 192   2          YLIMRT = 9830;               /* (149)    RIGHT YAW LIMIT (3.0 IN) */
 193   2          YLIMLT = -9830;              /* (150)    LEFT YAW LIMIT (3.0 IN) */
 194   2          KYSOFFR(0) = 0;              /* (151)    RIGHT YAW OFFSET 1 */
 195   2          KYSBRKR(0) = 0;              /* (161)    RIGHT YAW BREAKPOINT 1 */
 196   2          KYSPRINGR(0) = 49152;        /* (171)    RIGHT YAW SPRING COEFFICIENT 1 */
 197   2          KYSOFFL(0) = 0;              /* (181)    LEFT YAW OFFSET 1 */
 198   2          KYSBRKL(0) = 0;              /* (191)    LEFT YAW BREAKPOINT 1 */
 199   2          KYSPRINGL(0) = 49152;        /* (201)    LEFT YAW SPRING COEFFICIENT 1 */
 200   2          PTRIM = 0;                   /* (211)    PITCH TRIM INITIALLY ZERO */
 201   2          RTRIM = 0;                   /* (212)    ROLL TRIM INITIALLY ZERO */
 202   2          YTRIM = 0;                   /* (213)    YAW TRIM INITIALLY ZERO */
 203   2          KPLIMLEAD = 2;               /* (214)    PITCH LIMIT LEAD FACTOR */
 204   2          KRLIMLEAD = 2;               /* (215)    ROLL LIMIT LEAD FACTOR */
 205   2          KYLIMLEAD = 2;               /* (216)    YAW LIMIT LEAD FACTOR */
 206   2          KPBRKLEAD = 20480;           /* (217)    PITCH BREAKOUT LEAD FACTOR */
 207   2          KRBRKLEAD = 20480;           /* (218)    ROLL BREAKOUT LEAD FACTOR */
 208   2          KYBRKLEAD = 20480;           /* (219)    YAW BREAKOUT LEAD FACTOR */
 209   2          KPLIMF = 6;                  /* (220)    PITCH LIMIT FACTOR */
 210   2          KRLIMF = 6;                  /* (221)    ROLL LIMIT FACTOR */
 211   2          KYLIMF = 6;                  /* (222)    YAW LIMIT FACTOR */
 212   2          PTRIMRATE = 27;              /* (223)    PITCH TRIM RATE FACTOR */
 213   2          RTRIMRATE = 27;              /* (224)    ROLL TRIMRATE FACTOR */
 214   2          PTOT = 0;                    /*        PRESENT TOTAL PITCH FORCE */
 215   2          RTOT = 0;                    /*        PRESENT TOTAL ROLL FORCE */
 216   2          YTOT = 0;                    /*        PRESENT TOTAL YAW FORCE */
 217   2          PTOTOLD = 0;                 /*        LAST TOTAL PITCH FORCE */
 218   2          RTOTOLD = 0;                 /*        LAST TOTAL ROLL FORCE */
 219   2          YTOTOLD = 0;                 /*        LAST TOTAL YAW FORCE */
 220   2          KPLIMLAG = 2;                /*        PITCH LIMIT INTEGRATION FACTOR */
 221   2          KRLIMLAG = 2;                /*        ROLL LIMIT INTEGRATION FACTOR */
 222   2          KYLIMLAG = 2;                /*        YAW LIMIT INTEGRATION FACTOR */
 223   2          DO B1 = 0 TO 19;
 224   3              PLIMOLD (B1) = 0;
 225   3              RLIMOLD (B1) = 0;        /* INTEGRATION VARIABLES */
 226   3              YLIMOLD (B1) = 0;
 227   3              END;
 228   2          CALL FRAME;                  /* START FRAME RATE GENERATOR */
 229   2          OUTPUT(0E4H) = 0FFH;         /* FRAME-ACTIVE RESET */
```

TABLE 5.1 (Con't) PARAMETER NUMBER TABLE board as described in Section 5.2.5. The actual scan is conducted by simply calling SCAN, the assembly language routine described in Section 5.3. At this point the pitch equations are entered. The following section describes the operation of the pitch equations but this discussion can be extended to the roll axis and yaw axis equations as well. In fact the roll and yaw equations are duplicate copies of the pitch equations with the appropriate coefficients substituted. (The pitch coefficients can be recognized by a "P" in the variable name such as KPVISC. The roll and yaw equivalents are KRVISC and KYVISC respectively.) The yaw equations do not require a solution involving bobweight effects. Therefore, statements involving the bobweight effects which appear in the pitch and roll sections are deleted from the yaw section. Similarly, the yaw axis (rudder pedals) required no trim capability so the statements pertaining to trim rate are not included in the yaw processing but yaw trim effects are maintained so that the operator can adjust the trim from the console. These are the only differences between the processing of the three axes. Therefore, the description of the pitch processing is sufficient to describe all three axes.

5.4.4.3 Pitch Processing

The pitch processing is illustrated in Figure 5.3. The displacement direction is determined first so that it may be compared to the limit in the respective direction. If a limit is exceeded the appropriate limit calculations are made and the resulting value is output to the corresponding DAC. If the displacement is within limits it is checked for a value which might lie within the deadband, in which case the various dynamic stick calculations are bypassed. On the other hand, if the displacement is out of the deadband those terms are calculated and, in any event, the bobweight forces are calculated before the force is output to the DAC. Each of the specific areas of the pitch processing will be discussed in detail in the following paragraphs.

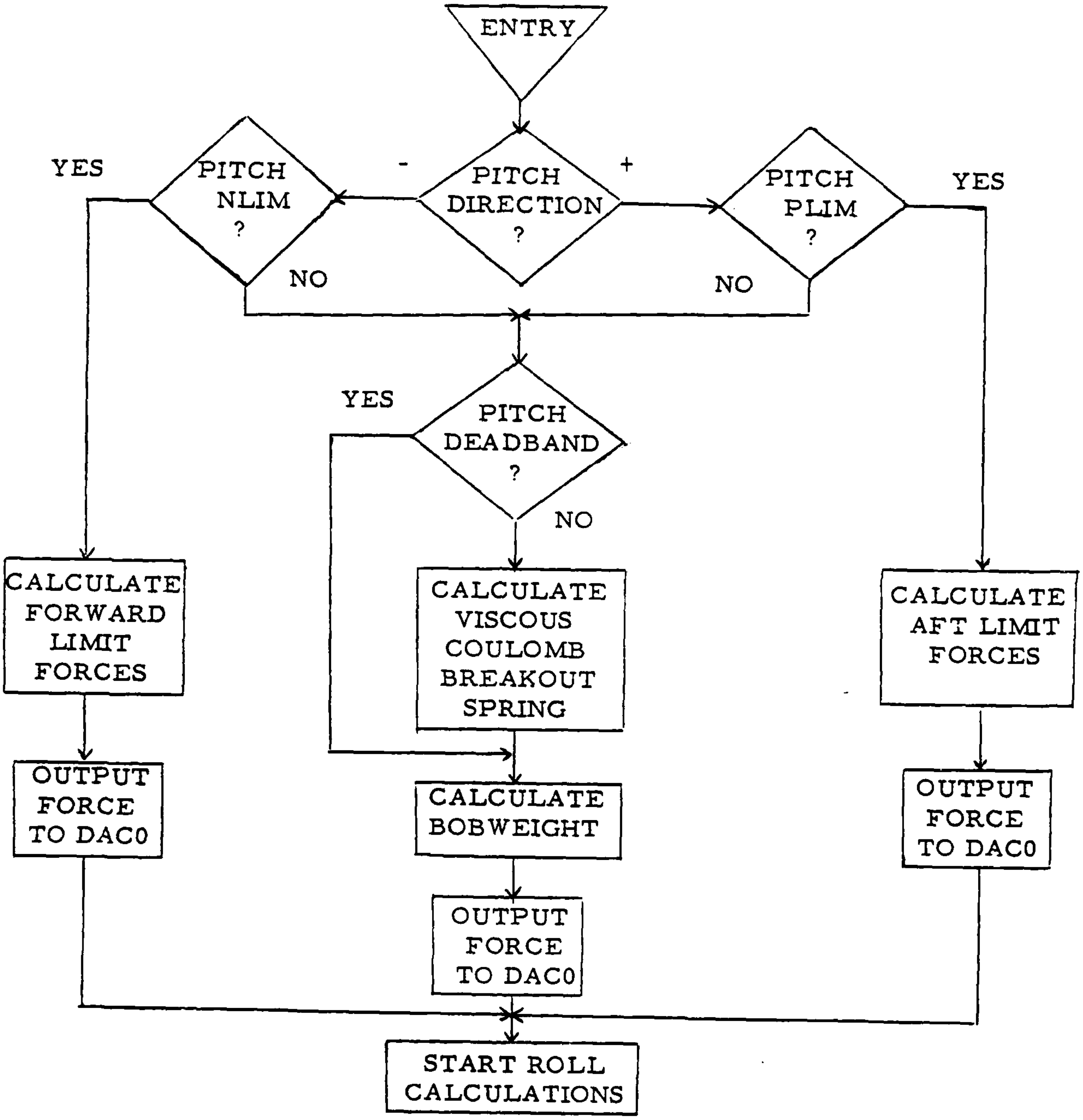


Figure 5.3 Pitch Processing Flowchart

5.4.4.3.1 Travel limit

The first operation is to evaluate the displacement for exceeding one of the travel limits. If the pitch is negative (meaning the stick is forward) control passes to the label NPITCH; if it is positive the evaluation is made in sequence. It should be mentioned here that "negative" means a hexadecimal value greater than 7FFF which is consistant with the offset binary (2's complement) convention set up on both analog boards. In either case the pitch is compared with the appropriate limit (PLIMAFT or PLIMFOR) and if it is determined to be within limits control is passed to the label PITCH$DEADBAND for further processing. If the limit has been exceeded then the difference (the amount in excess) is multiplied by a gain factor KPLIMF. The result of this multiplication is tested for an overflow (a value greater than the positive limit). In this case an overflow would occur during the subsequent integration if the result was greater than 3FFF, half the offset binary limit. If an overflow is detected then the maximum opposing force is commanded (8000 hex for a positive displacement, 7FFF hex for a negative displacement) and command passes to the roll equations. If no overflow is detected then command passes to the procedure PITCH$FORCE which applies the compensation to the force value. The result of the above multiplication (the amplified displacement excess) is passed to this procedure with the proper sign (negative for aft displacement, positive for forward displacement) through the general address-type variable A1.

The PITCH$FORCE procedure provides a lag term by integrating (adding) the present value with the previous value and it provides a lead term through a subsequent procedure called PLEAD. The intergration routine can accomodate up to 20 terms but the present configuration only requires 2 which is governed by the value of KPLIMLAG. The lead factor is added by employing the pitch velocity, PDOT, multiplied by a gain factor KPLIMLEAD. However, due to the fact that the multiplication must be performed with positive numbers the velocity is first tested for polarity and the absolute value of PDOT is loaded into the operand register. The multiplication is performed and a check for an overflow is again made. The resultant lead term is added to the integrated lag term and the pre-limit value of the pitch force, PTOT. PTOT is included so that the spring and other linear forces are incorporated in the overall force value. A final check for overflows is made before the force value is written out to the pitch output channel, DAC0.

5.4.4.3.2 Deadband

If the pitch displacement is not out of the travel limits, control is passed to the label PITCH$DEADBAND. At this point the trim signal PTRIM$SIG is tested to determine whether the pilot is commanding trim and, if so, the direction of the command. A forward trim command is designated by a +10 volts on analog input channel 10; an aftward trim command by a -10 volts. The discrimination of this signal is made above +2.5 volts (8192 decimal) and below -2.5 volts (57344 decimal). The trim is thereby shifted at a rate of approximately one inch/second by adding (subtracting) the pitch trim rate factor, PTRIMRATE, with the previous trim value. It must be noted that PTRIM is positive aft, negative forward. Processing continues, regardless of the trim processing, by initializing the total pitch force variable PTOT to zero. A check is made to determine whether the stick is within the deadband. Trim is incorporated with the pitch value in this determination since the deadband travels with the trim setting. If the net value is positive a comparison is made with the aft deadband limit, PDBANDA. If it is negative, the comparison is made with the forward limit PDBANDF. If the stick is determined to be within the deadband there is no need to calculate any of the linear parameters so control is passed to the bobweight equations. On the other hand if the stick is in the linear region processing progresses sequentially.

5.4.4.3.3 Spring and breakout forces

If the stick is past a deadband limit, the absolute value of this net linear distance is used to determine the spring and breakout forces. That is, the processing is divided at this point based on this net value so that the absolute value will be used and sothat the proper spring coefficients are used. The net value is used to also determine which interval of the non-linear spring is to be selected. A comparison is made with the spring breakpoints, the array KPSBRKF (or KPSBRKA), and the appropriate spring constant is selected from the array KPSPRINGF (or KPSPRINGA). This net distance is multiplied by this spring constant and the appropriate offset element from the array KPSOFFF (or KPSOFFA) is added to the total force PTOT. In addition, the breakout force must be added to PTOT before moving on to the other terms.

Breakout was determined to require lead and lag compensation around the point of the break. The lag term is incorporated by letting the value of the breakout force be equal to twice the net displacement distance past the deadband limit until this force reaches that of the breakout parameter KPBRAK. The lead term is incorporated to provide a snapping sensation at the break point. This is done by adding the velocity multiplied by a gain factor called KPBRK-LEAD to the total force PTOT. However, this lead term is only added if the stick is within 0.1 inches (328 decimal units) of the breakpoint. Processing subsequently passes to the friction equations regardless of the processing path taken through the spring and breakout forces.

5.4.4.3.4 Viscous and coulomb friction forces

The viscous friction forces are calculated by simply multiplying the abosulte value of the velocity PDOT by the viscous coefficient KPVISC. The coulomb funtion, however, required a small lag term initially. Therefore the force value (variable A2 at this point) is set equal to the velocity until the value reaches the flat coulomb force value KPCOUL. Both force values are added to the force total PTOT.

5.4.4.3.5 Bobweight forces

There are two bobweight force terms to consider. The first is the term due to the normal acceleration of the aircraft which is passed to the microcomputer from the host computer and is named NZ. This NZ term is simply multiplied by the appropriate coefficient KNZ to produce the resultant force which is added to PTOT. Similarly, QDOT, the angular airframe acceleration from the host, is multiplied by its respective coefficient KQDOT and added to PTOT.

5.4.4.3.6 Force integration and output

An empirical study revealed that the force value in the linear stick region from the deadband limit to the travel limit should be smoothed by integrating. Therefore, the value which is finally written to the DAC is the average of the present calculated value and the previous value. This averaging is performed by dividing the present value by 2 (shift right one bit) and adding this value to the previous halved value. The resultant value is written out to DAC0 and the present halved value is saved for the next frame. At this point, processing passes on to the roll equations and subsequently to the yaw equations.

5.4.4.3.7 End-of-frame routines

After the yaw equations are completed, a series of steps are taken which terminate the processing for the present frame. First, Bit 1 at J1 is brought to a low state which, when viewed on an oscilloscope signifies the end-of-frame. Secondly, the remaining count is read from the frame generator counter to provide an exact value for frame time remaining. This operation was used during development under ICE-80 control and has no function for the real-time system. Lastly, the wait loop is entered until the end of frame interrupt occurs. (See Section 5.4.2.) At this point control passes back to the beginning of the simulator loop at SIMLOOP unless a console request is pending from the operator. If such a request is pending the actions described in Section 6.6.3.2 will be taken.

SECTION 6
COMPUTER PROGRAM SYSTEM USERS GUIDE

The hardware items which were procured for this project were strictly off-the-shelf components. These components were capable of being modified to a certain extent. However, the generalized nature of this equipment was not affected by any of the modifications which were performed. The equipment was designed to be largely software configured. This section, therefore, describes the software which is necessary to set up the hardware components for this project, perform utility functions, and conduct the real-time simulation.

6.1 GENERAL INFORMATION

The entire control loader software was written in the Intel high-order microcomputer language called PLM (PL/M-80), with the exception of the analog-to-digital conversion routine which was performed in assembly language. The programs were structured into a utility module, a scan module, and a simulator module. As the name implies the utility module provided system support in the form of calibration and test routines, system interface procedures, and system initialization. The scan module causes twelve of the available sixteen A-to-D converter channels to be scanned sequentially and the converted values to be placed in the appropriate RAM locations for later use. The simulator module contains the frame rate generator, simulator initialization routines, and the simulator equations themselves. The executable machine code was placed in PROM (programmable read only memory). The variable data and equation coefficients are placed in RAM (Random Access Memory).

6.2 SOFTWARE DEVELOPMENT

The software, including both the PLM and assembly language modules, was developed on an Intel MDS (Microcomputer Development System). The complete system development required the following system components:

- MDS with 64K of memory
- Dual density floppy disk system (dual drive)
- System console (TTY)
- PROM programmer
- In-circuit-emulator (ICE-80)
- Line printer

The MDS supports PLM with a coupler called PLM80 and an assembler called ASM80. (The reader is referred to the appropriate language and/or operator's manual in Volume 2 of the Computer Program Documentation (CPD).) Furthermore, the MDS links the various modules together (including the PLM library module) with an Intel program LINK. The linked program is subsequently located in the appropriate memory addresses with another Intel program LOCATE. (See ISIS II User's Guide.) The executive code was placed in PROM by employing Intel's program UPM (see Universal PROM Mapper Operator's Manual). During the development phase, the memory and CPU of the System 80/20 was emulated by the MDS through the use of Intel's ICE-80 In-Circuit-Emulator and its associated software program called ICE80. (See ICE-80 Hardware Reference Manual and Operator's Manual.)

6.3 SIMULATOR EQUATIONS

A great number of tasks are accomplished in the SIMULATOR procedure and its associated procedures. However, before discussing the specific mechanisms involved, a general discussion is needed in order to relate the approach taken to the overall problem. Similarly it is necessary to discuss equation scaling and to relate the actual microcomputer values associated with each parameter.

Each of the individual control loading force components in some way models an element of the aircraft control system. These forces are, in general, additive as shown in Figure 6. 1 even though they may be the result of non-linear processes. However the total solution requires that these forces be logically connected to accurately represent the fully integrated control system.

As an example of the microcomputer's flexibility, consider the forces generated for a single axis (e. g. pitch control) which has a deadband due to rigging slack at the control stick. The pilot would therefore experience nearly zero force within the deadband which is described as follows:

$$\begin{aligned} F_{TOT} &= [F_{SP} + F_{VI} + F_{CO} + F_{BR}]U(DB) \\ &\quad + F_{TR} + F_{VEL} + F_{AC} \end{aligned} \tag{6. 1}$$

where the above terms stand for total, spring, viscous, coulomb, breakout travel limit, velocity limit and aircraft related forces respectively. The unit operator "U" indicates that the quantities within the brackets are nil in the deadband.

As an alternate example, consider a case where the rigging slack is remote from the stick. The coulomb friction due to pully drag may then be placed outside the influence of the deadband with the resultant force equation

$$\begin{aligned} F_{TOT} &= [F_{SP} + F_{VI} + F_{BR}]U(DB) \\ &\quad + F_{CO} + F_{TR} + F_{VEL} + F_{AC} \end{aligned} \tag{6.2}$$

These two examples represent wiring differences in an analog system but only software differences in a digital system, an important consideration in a research or development environment. For the A-10 model, the deadband is very small but follows the format of Equation 6. 1. The

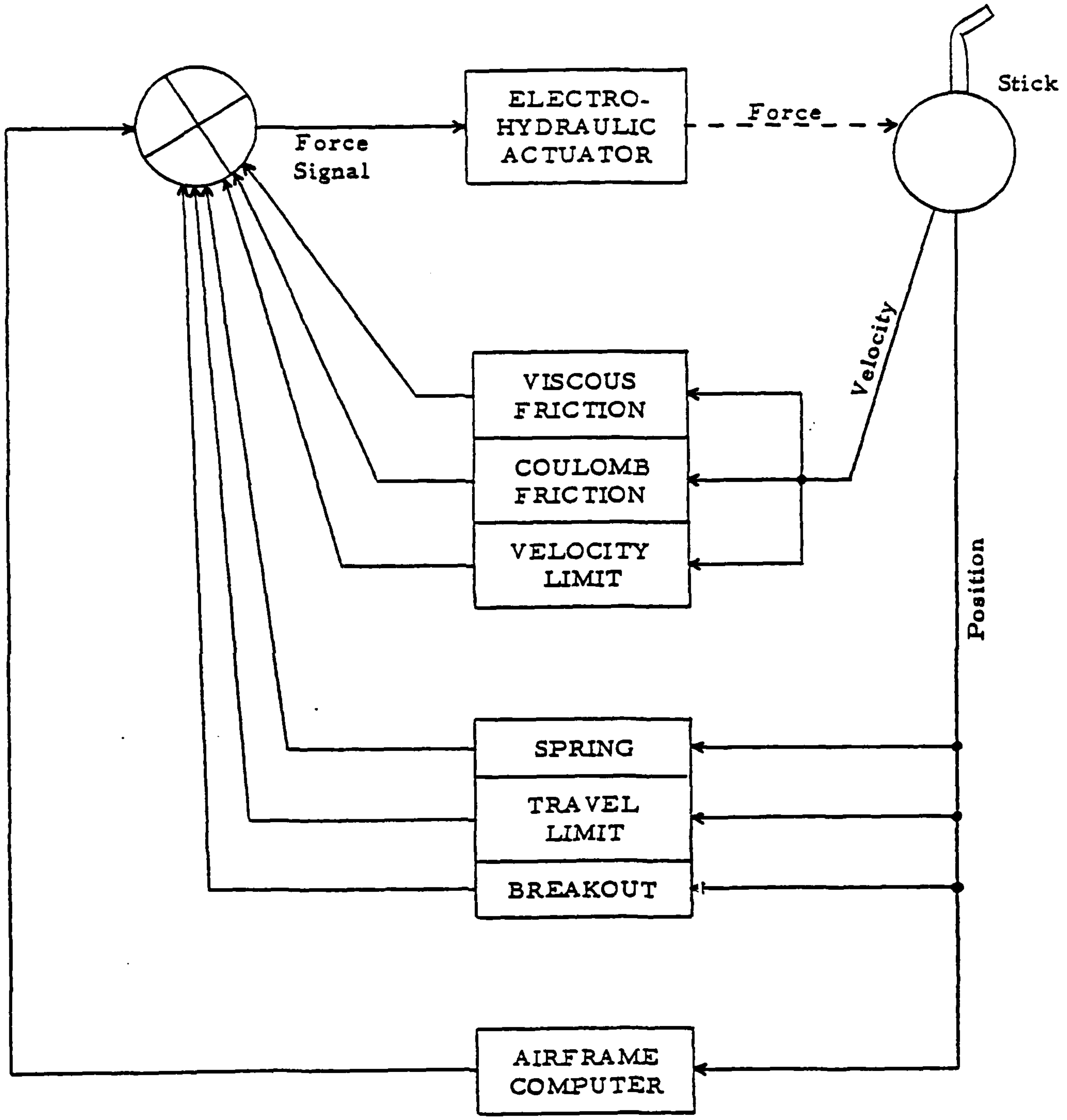


Figure 6.1 Force Component Diagram - Single Axis

complete pitch force equation (from Section 4.5.1) then becomes

$$\begin{aligned} F_{TOT_p} = -[&\underbrace{F_{SP_p}}_{\text{spring}} + \underbrace{F_{BR_p}}_{\text{breakout}} + \underbrace{F_{CO_p}}_{\text{coulomb}} + \underbrace{.64\,\dot{X}_p]}_{\text{viscous}} \\ &\underbrace{U(DB)_p}_{\text{deadband}} + \underbrace{F_{TR_p}}_{\text{travel}} - \underbrace{(3N_Z + .026\dot{q})}_{\text{bobweight (aircraft)}} \end{aligned} \tag{6.3}$$

where $\dot{X}$ is the velocity in inches-per-second, N_Z and $\dot{q}$ are the normal and angular accelerations from the airframe computer which produce the control system bobweight effects. The force is calculated in pounds and, of course, requires a sign change to produce the required opposing force. Equation 6.3 is obviously stylized and requires special handling due to the non-linearities involved.

In order to handle the non-linearities of such a system, the frequency response of the control loader must be considered. The nominal 20 frames-per-second for digital simulators is usually sufficient to make the pilot believe that he is flying in a parallel, analog world as evidenced by his visual displays. However, the pilot's tactile mechanism is capable of much higher frequency response. Empirical studies conducted on the McFadden control loader revealed frequency components of 1000 Hz and higher. As one might expect, these components are experienced at the breakout and travel limits where forces suddenly change. To satisfy the ground rules of information theory put forth by Shannon, the microcomputer should ideally be framing at a 5000 Hz rate or better. However, this project demonstrated that quite acceptable results can be had at 120 frames-per-second by judiciously choosing compensation schemes. That is, by optimizing lead, lag and gain coefficients in the individual component force calculations relatively sharp breakouts and stops were obtained.

Various techniques were attempted to achieve the proper compensation, including a Tustin recursion method to approximate the desired transfer function. However, the final product was a result of an educated cut-and-try effort. For example consider the travel limit force component which can be expressed mathematically as a recursion relationship

$$F_{TR_p} = K_p[X_p(N) + X_p(N-1)] + K_{L_p}\dot{X}_p \qquad (6.4)$$

In the digital implementation, the needed lag term is obtained by employing both the present frame value for travel limit displacement $X_p(N)$ and the previous frame value $X_p(N-1)$. The gain is controlled by the value of K_p and the needed lead term is obtained from the available velocity signal X_p which is modified by the constant K_{L_p}. The unscaled magnitudes of K_T and K_{L_p} and the number of terms in the recursion portion of the lag component were determined empirically with the final values set at $K_p = 6$ and $K_{L_p} = 2$. The breakout term of Equation 6.3 required a slight lag compensation augmented by a lead term ($K_{B_p} = 0.3$) and was empirically structured as

$$\begin{aligned} F_{BR_p} &= 2X_p + K_{B_p}\dot{X}_p \text{ for } 2\left|X_p\right| < 2.5 \\ &= 2.5 \text{ sign } (X_p) \text{ for } 2\left|X_p\right| \geq 2.5 \end{aligned} \qquad (6.5)$$

Similarly, the coulomb friction term required compensation to offset a tendency to "dither" because of its dependency on the sign of the velocity. In this case a simple lag was implemented by employing the velocity value as follows:

$$\begin{aligned} F_{CO_p} &= \dot{X}_p && \text{for } \left|\dot{X}_p\right| < 1.5 \\ &= 1.5 \text{ sign } (\dot{X}_p) && \text{for } \left|\dot{X}_p\right| \geq 1.5 \end{aligned} \qquad (6.6)$$

The equations executed by the microcomputer for each of the respective axes are summarized as follows:

<u>Pitch</u>

Refer to equations 6.3, 6.4, 6.5, and 6.6 above.

Roll

$$F_{TOT_R} = -[\underbrace{4.5X_R}_{\text{spring}} + \underbrace{F_{BR_R}}_{\text{breakout}} + \underbrace{F_{CO_R}}_{\text{coulomb}} + \underbrace{.06\dot{X}_R}_{\text{viscous}}]U(DB)_R \qquad (6.7)$$

$$- \underbrace{F_{TR_R}}_{\text{travel}} - \underbrace{(5N_y + .008\dot{p})}_{\text{limb bobweight}}$$

$$F_{BR_R} = 2X_R + K_{B_R}\dot{X} \text{ for } 2\left|X_R\right| < 2.0 \qquad (6.8)$$

$$= 2.0 \text{ sign } (X_R) \text{ for } 2 \left|X_R\right| \geq 2.0$$

$$F_{CO_R} = \dot{X}_R \text{ for } \left|\dot{X}_R\right| < 1.5 \qquad (6.9)$$

$$= 1.5 \text{ sign } (\dot{X}_R) \text{ for } \left|\dot{X}_R\right| \geq 1.5$$

$$F_{TR_R} = K_R [X_R(N) + X_R(N-1)] + K_{L_R}\dot{X}_R \qquad (6.10)$$

$$\text{where } K_R = 6 \text{ and } K_{L_R} = 2$$

Yaw

$$F_{TOT_Y} = -[15X_Y + F_{BR_Y} + F_{CO_Y} + .905\dot{X}_Y]U(DB)_Y - F_{TR_Y} \qquad (6.11)$$

$$F_{BR_Y} = 2X_Y + K_{B_Y}X_Y \text{ for } 2\left|X_Y\right| < 5.0 \qquad (6.12)$$

$$= 5.0 \text{ sign } (X_Y) \text{ for } 2 \left|X_Y\right| \geq 5.0$$

$$F_{CO_Y} = \dot{X}_Y \text{ for } \left|\dot{X}_Y\right| < 4.25 \qquad (6.13)$$

$$= 4.25 \text{ sign } (\dot{X}_Y) \text{ for } \left|\dot{X}_Y\right| \geq 4.25$$

$$F_{TR_Y} = K_Y[X_Y(N) + X_Y(N-1)] + K_{L_Y}X_Y \qquad (6.14)$$

$$\text{where } K_Y = 6 \text{ and } K_{L_Y} = 2$$

6.4 EQUATION SCALING

The first step in scaling the equations was to properly determine sign conventions. Using the McFadden control loader it was determined that the pitch, roll and yaw axes are positive in the aft, right, and right directions respectively. That is, the position, velocity and force vectors for the pitch axis (for example) is positive aft. (Refer to Table 4. 1 and Figure 4. 1.) To further clarify this point consider a spring force acting on the pitch stick axis. A positive (aft) displacement will result in a negative (forward) force.

The second step in the scaling process was to determine the input and output sensitivities and then formulate these sensitivities and the equation coefficients to obtain the microcomputer parameters. On the input side, the position and velocity values (for all axes) are related in inches and inches/second, respectively. The force output is related in pounds. The microcomputer converts voltages to/from digital values through its Analog-to-Digital Converter (ADC) and its Digital-to-Analog Converter (DAC). Both of these devices have a 12-bit, bilateral (± 10 v) resolution. However, since the 8080 microcomputer operates on multiples of 8 bits, the converter bits are placed in the most significant positions of a 16-bit word. Therefore, the full range of 20 volts (± 10 v) can be represented in 15 bits or 2^{16} (65,536) counts. In other words, the A/D conversion factor is $\frac{65536}{20\text{ v}}$ or 3276.8 $\frac{\text{COUNTS}}{\text{V}}$. The control loader sensitivity terms, therefore, can be expressed as digital values and these are shown in Table 6. 1.

In order to perform the calculations in the microcomputer, each equation coefficient must be conditioned by the proper input and output sensitivity values. For example, if the viscous friction constant was .64 lbs/in/sec, then the digital value which represents the force resulting from a digital velocity PDOT is

TABLE 6.1 INPUT/OUTPUT SENSITIVITIES

	Control Loader Units	Voltage Sensitivity	Digital Sensitivity
Pitch			
Displacement	inches	$\frac{1\ \mathrm{v}}{1\ \mathrm{in}}$	3277 $\frac{\mathrm{counts}}{\mathrm{in}}$
Velocity	$\frac{\mathrm{in}}{\mathrm{sec}}$	$\frac{1\ \mathrm{v}}{5\ \mathrm{in/sec}}$	655 $\frac{\mathrm{counts}}{\mathrm{in/sec}}$
Norm. Acc.	G's	$\frac{1\ \mathrm{v}}{1\ \mathrm{G}}$	3277 $\frac{\mathrm{counts}}{\mathrm{G}}$
Rot. Acc.	$\frac{\mathrm{deg}}{\mathrm{sec}^2}$	$\frac{1\ \mathrm{v}}{4\ \mathrm{deg/sec}^2}$	819 $\frac{\mathrm{counts}}{\mathrm{deg/sec}^2}$
Force	lb	$\frac{1\ \mathrm{v}}{15\ \mathrm{lb}}$	218 $\frac{\mathrm{counts}}{\mathrm{lb}}$
Roll			
Displacement	inches	$\frac{1\ \mathrm{v}}{1\ \mathrm{in}}$	3277 $\frac{\mathrm{counts}}{\mathrm{in}}$
Velocity	$\frac{\mathrm{in}}{\mathrm{sec}}$	$\frac{1\ \mathrm{v}}{6\ \mathrm{in/sec}}$	546 $\frac{\mathrm{counts}}{\mathrm{in/sec}}$
Norm. Acc.	G's	$\frac{1\ \mathrm{v}}{1\ \mathrm{G}}$	3277 $\frac{\mathrm{counts}}{\mathrm{G}}$
Rot. Acc.	$\frac{\mathrm{deg}}{\mathrm{sec}^2}$	$\frac{1\ \mathrm{v}}{2\ \mathrm{deg/sec}^2}$	1638 $\frac{\mathrm{counts}}{\mathrm{deg/sec}^2}$
Force	lb	$\frac{1\ \mathrm{v}}{10\ \mathrm{lb}}$	328 $\frac{\mathrm{counts}}{\mathrm{lb}}$
Yaw			
Displacement	inches	$\frac{1\ \mathrm{v}}{1\ \mathrm{in}}$	3277 $\frac{\mathrm{counts}}{\mathrm{in}}$
Velocity	$\frac{\mathrm{in}}{\mathrm{sec}}$	$\frac{1\ \mathrm{v}}{5\ \mathrm{in/sec}}$	655 $\frac{\mathrm{counts}}{\mathrm{in/sec}}$
Force	lb	$\frac{1\ \mathrm{v}}{20\ \mathrm{lb}}$	164 $\frac{\mathrm{counts}}{\mathrm{lb}}$

$$F_{VI} = - \frac{.64 \frac{LB}{IN/SEC} \cdot 218 \frac{COUNTS}{LB}}{655 \frac{COUNTS}{IN/SEC}} \cdot \text{PDOT counts}$$

$$= -.2133 \text{ PDOT counts (see Table 6.2)}$$

In other words, the digital value PDOT would be read from the A/D converter and then multiplied by the dimensionless microcomputer coefficient, 0.2133. The resulting value is then complimented to give the proper sign and thus produce the digital component due to spring force F_{VI}. F_{VI} may then be added with other components before being written to the Digital-to-Analog Converter (DAC).

To perform the above multiplication ordinarily requires that the operation be carried out in floating point format. The input value PDOT must be converted to floating point, the multiplication performed, and the result converted back to fixed point (integer) format. This is a very time consuming process and, therefore, a technique was devised to produce the desired product by integer multiplication. The SBC 310 High Speed Math Board produces a 32-bit integer product from two 16-bit integer operands. This 32-bit product can be considered to consist of a low order 16-bit word and a high order 16-bit word which as a weight 2^{16} (65536) times larger than the low order word. The control loading calculations must always result in a 16-bit value since the DAC only accepts 16-bits. Ordinarily, the desired result of a multiplication involving an operand (PDOT) which is less than 65536 and a coefficient (e.g. 0.2133) would itself be less than 65536 and it would reside in the low order word of the SBC 310. However, if the coefficient is scaled up by a factor of 65536 then the desired result will reside in the high order word. The microcomputer coefficient in this case is

$$KPVISC = 0.2133 \times 65536 = 13981$$

The equation implemented in the microcomputer is then

$$F_{VI} = - \text{HIGH (KPVISC} \cdot \text{PDOT) COUNTS}$$

where "HIGH" refers to employing the high order 16-bit word of the 32-bit result of the SBC 310.

Figure 6.2 illustrates this integer multiplication algorithm which is used throughout the simulator processing. The coefficient KPVISC is loaded in the lower sixteen bits of operand register 2 on the high speed math board. Likewise, the absolute value of the velocity PDOT as read from the ADC is placed in the lower sixteen bits of operand register 1. The integer multiplication operation "TIMES" is invoked (via the PERFORM procedure) and the 32-bit product of the unsigned multiplication (the reason for using the absolute value) is placed in operand register 1. Since KPVISC was scaled up by 2^{16} the properly scaled product is also scaled up by 2^{16} and therefore resides in the high order half of register 1.

The net result of this technique is a very distinct speed advantage: 200 microseconds for a complete integer multiplication versus 1000 microseconds for a complete floating-point multiplication (float/fix conversion included). All calculations are now carried out in the microcomputer in integer format. Table 6.2 contains the values of pertinent coefficients for microcomputer processing. The other terms (without scaled values) are parameters in the total force output but are not coefficients for multiplication. For example, if the pitch coulomb value of 1.5 pounds is to be added to the total force, then the integer value of 328 ($1.5 \text{ LB} \times 218 \frac{\text{COUNTS}}{\text{LB}}$) is added to the integer force value which eventually is written to the output DAC.

The springforce modeled by the microcomputer can be either linear or non-linear. In the case of the A-10 the pitch axis spring is the only non-linear force producer. The other axes, roll and yaw, are linear. Furthermore, the pitch axis is also non-symmetrical in the fore and aft forces which it produces.

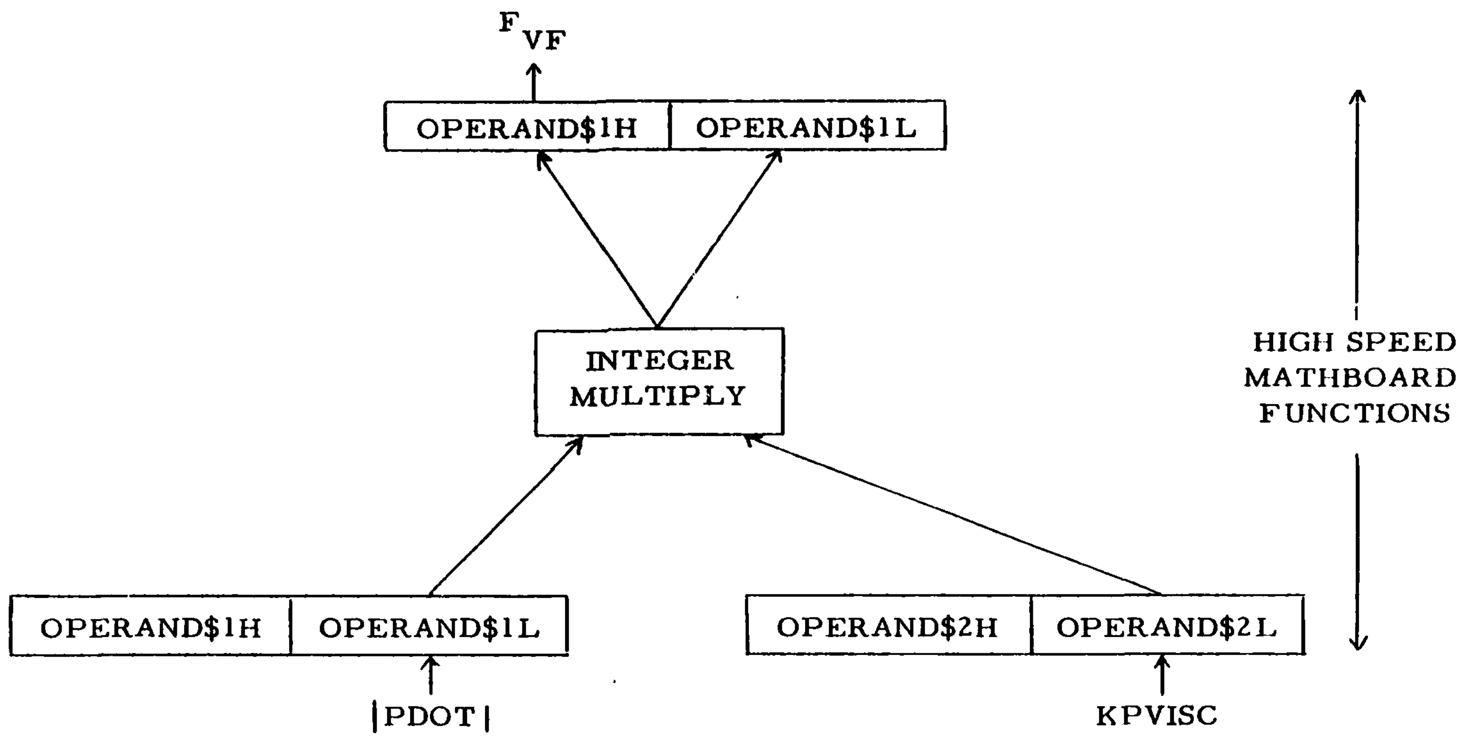


Figure 6.2 Integer Multiplication Algorithm

TABLE 6.2 MICROCOMPUTER COEFFICIENTS

TERM	VALUE	SCALED VALUE	INTEGER VALUE (X 64K)
KPVISC	0.64 lb/in/sec	.2133	13981
KPCOUL	1.5 lb	-	328
KPBRAK	2.0 lb	-	437
PLIMAFT	5.7 in	-	18678
PLIMFOR	2.7 in	-	8847
KNZ	3.0 lb/G	.200	13107
KQDOT	0.026 lb/deg/sec^2	.00693	454
KRVISC	0.06 lb/in/sec	.036	2359
KRCOUL	1.5 lb	-	492
KRBRAK	2.0 lb	-	655
KRSPRING	4.5 lb/in	.45	29491
RLIMRT	3.1 in	-	10158
RLIMLT	3.1 in	-	10158
KRVLIM	6.5 in/sec	-	3550
KNY	5.0 lb/G	.50	32768
KPPDOT	0.008 lb/deg/sec^2	.0016	105
KYVISC	0.905 lb/in/sec	.2263	14827
KYCOUL	4.25 lb	-	696
KYBRAK	5.0 lb	-	819
KYSPRING	15.0 lb/in	.75	49152
YLIMRT	3.5 in	-	11468
YLIMLT	3.5 in	-	11468

The general procedure which is followed to produce the spring force values is that of calculating a straight line if the form

$$y = mx + b$$

where "m" is the spring coefficient (slope), "x" is the displacement, and "b" is the offset force. For a totally linear spring the offset "b" is zero. Non-linear springs, on the other hand, are calculated in a piecewise manner where the first segment (nearest the origin) will have an *offset* "b" equal to zero and all other segments will have a non-zero offset. Figure 6.3 illustrates the piecewise approximation of the pitch spring force curve. The various breakpoints and offsets are shown on the curve and the spring coefficients were calculated from the slope of each segment. The decimal values reflect the real world parameter and the integers are the values which were used in the microcomputer calculations. Table 6.3 summarizes these microcomputer parameters.

TABLE 6.3 PITCH SPRING PARAMETERS

	SEGMENTS				
Name	1	2	3	4	5
KPSOFFA	0	851	1373		
KPSBRKA	0	4260	11469		
KPSPRINGA	23522	10427	7445		
KPSOFFF				0	341
KPSBRKF				0	4915
KPSPRINGF				21840	17300

These parameters were placed in arrays corresponding to the names at the left and the array length obsiously corresponds to the number of segments in the curve. As an example of a spring force calculation within the microcomputer consider a situation where the stick is displaced 2 inches aft. The converted digital value PITCH would be 6554 (2 IN X 3277 $\frac{\text{COUNTS}}{\text{IN}}$). The microcomputer determines that this value lies in segment number 2 on the spring force curve. The final spring force

Slopes	Lb/In	KPSPRING (X 65K)
1	5.38	23522
2	2.39	10427
3	1.70	7445
4	5.00	25199
5	3.96	14830

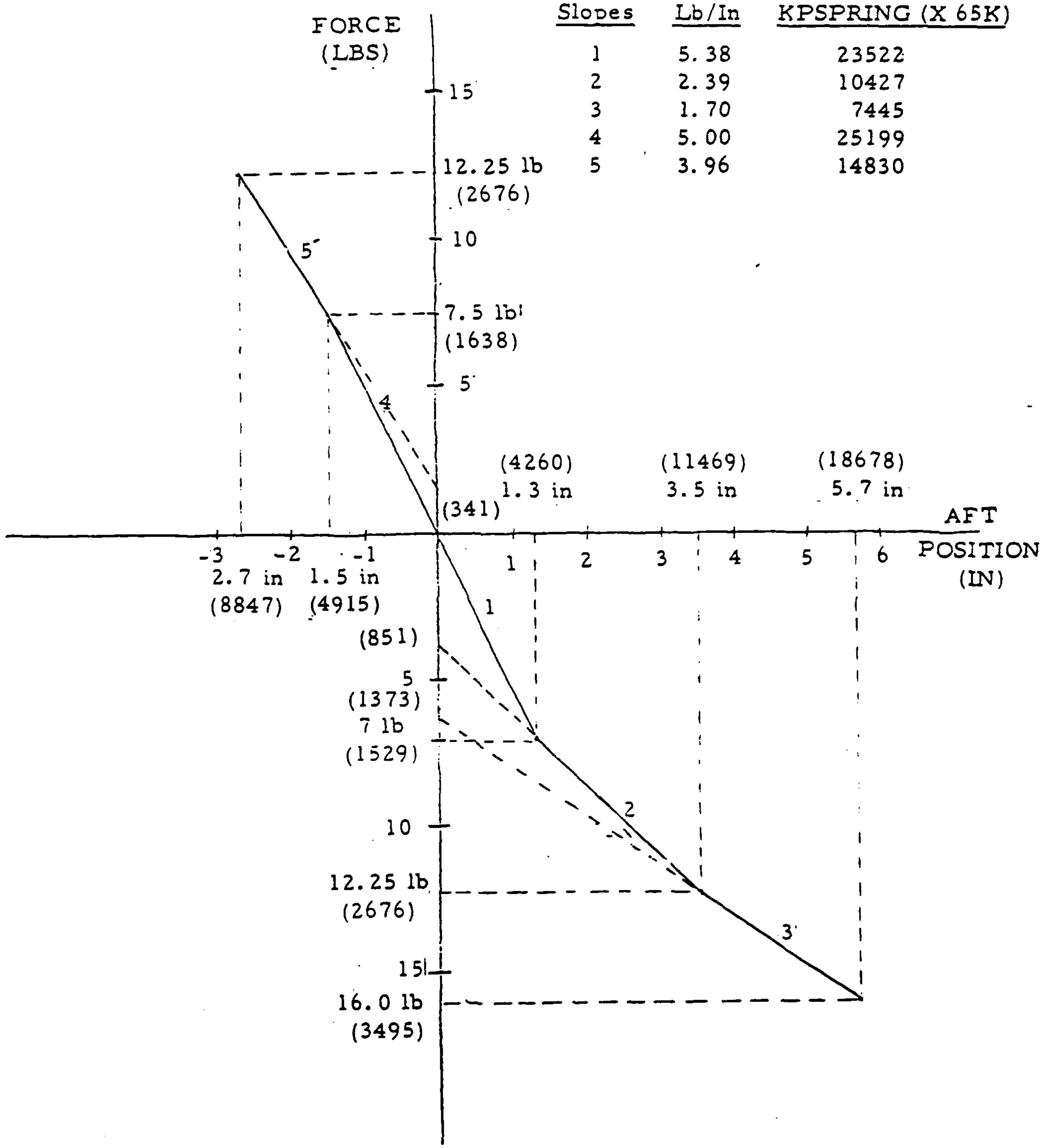


Figure 6.3 Pitch Spring

digital value is calculated by integer multiplying the PITCH value by the slope KPSPRINGA(2) and adding the offset KPSOFFA(2). Thus

$$\begin{aligned} F_{SP} &= \text{HIGH}\ (\text{KPSPRINGA}(2) \cdot \text{PITCH}) + \text{KPSOFFA}(2) \\ &= \text{HIGH}\ (10427 \cdot 6554) + 851 \\ &= 1043 + 851 = 1894\ \text{COUNTS}\ (8.69\ \text{LBS}) \end{aligned}$$

6.5 EQUATION IMPLEMENTATION

The equations of Section 6.3 are implemented in the final software in a form such that the microcomputer and the high speed mathematics board perform unsigned (positive) integer multiplication of 16-bit quantities. This operation is carried out by specifically loading the operands into the mathematics board and retrieving the results as described in Section 6.4. Because of this process, the equations which are executed cannot be expressed in FORTRAN-like statements. However, it may perhaps be helpful to express the mathematical equations of Section 6.4 in a standard format using the mnemomics of the PLM software. Therefore, the following paragraphs are a summary of these microcomputer program "pseudo-statements." This summary should be reviewed in conjunction with the detailed explanation in Section 5.4.4, Simulator Processing.

6.5.1 Pitch Equations

The digital value of the pitch force is written to the appropriate analog line through digital-to-analog converter DAC0. Therefore, DAC0 is equated to either the limit force or the linear force, depending on the magnitude of the pitch displacement (see Figure 5.3). The pitch limit force is composed of a linear term, a lag term and a lead term and can be expressed in the aft direction as:

$$\begin{aligned} \text{DAC0} = &- \text{KPLIMF}\ (\text{PITCH}-\text{PLIMAFT}) + \text{PLIMOLD}(1) \\ &+ \text{KPLIMLEAD}\ (\text{PDOT}) \end{aligned}$$

Subsequently for the next frame

PLIMOLD (1) = - KPLIMF (PITCH-PLIMAFT)

For the linear portion of the force range, the following series of statements express the calculations which take place to incorporate each of the modeled parameters:

Spring and Breakout

PTOT = (PITCH+PTRIM) KPSPRINGA(B2) + KPSOFFA(B2)

+ KPBRAK

where B2 is the spring segment number.

Breakout Lead

PTOT = PTOT + KPBRKLEAD (PDOT)

Bobweight Effects

PTOT = PTOT + KNZ (NZ)

PTOT = PTOT + KQDOT (QDOT)

Integration and Output

DAC0 = PTOTOLD + PTOT/2

PTOTOLD = PTOT/2

The above equations would be modified for a forward stick displacement by appropriately replacing PLIMAFT, KPSPRINGA, and KPSOFFA with PLIMFOR, KPSPRINGF, and KPSOFFF.

6.5.2 Roll Equations

The analog value of the roll forces is produced by DAC1 and, similar to the pitch equations, the roll equations can be expressed for the limit force as

DAC1 = - KPLIMF (ROLL-RLIMRT) + RLIMOLD(1)

+ KRLIMLEAD(RDOT)

RLIMOLD(1) = - KRLIMF (ROLL-RLIMRT)

For the linear portion of the force range, the following series of statements express the calculations which take place to incorporate each of the modeled parameters

Spring and Breakout

RTOT = (ROLL+RTRIM) KRSPRINGR(B2) + KRSOFFR(B2)

+ KRBRAK

where B2 is the spring segment number

Breakout Lead

RTOT = RTOR + KRBRKLEAD (RDOT)

Viscous and Coulomb Friction

RTOT = RTOT + KRVISC (RDOT) + KRCOUL

Bobweight Effects

RTOT = RTOT + KNY (NY)

RTOT = RTOT + KPPDOT (PPDOT)

Integration and Output

DAC1 = RTOTOLD + RTOT/2

RTOTOLD = RTOT/2

The above equations would be modified for a left stick displacement by appropriately replacing RLIMRT, KRSPRINGR and KRSOFFR with RLIMLT, KRSPRINGL, and KRSOFFL.

6.5.3 Yaw Equations

The analog value of the yaw force is produced by DAC2 and, similar to the pitch and roll equations, the yaw equations can be expressed for the limit force as

DAC2 = KYLIMF (YAW-YLIMRT) + YLIMOLD(1) + KYLIMLEAD (YDOT)
YLIMOLD = - KYLIMF (YAW-YLIMRT)

For the linear portion of the force range, the following series of statements express the calculations which take place to incorporate each of the modeled parameters.

Spring and Breakout

YTOT = (YAW+YTRIM) KYSPRINGR(B2) + KYSOFFR(B2)
+ KYBRAK

where B2 is the spring segment number.

Breakout Lead

YTOT = YTOT + KYBRKLEAD (YDOT)

Viscous and Coulomb Friction

YTOT = YTOT + KYVISC (YDOT) + KYCOUL

Integration and Output

DAC2 = YTOTOLD + YTOT/2
YTOTOLD = YTOT/2

The above equations would be modified for a left rudder displacement by appropriately replacing YLIMRT, KYSPRINGR, and KYSOFFR with YLIMLT, KYSPRINGL, and KYSOFFL.

6.6 HARDWARE INTERFACE DESCRIPTION

Interfacing the microcomputer with the control loader and the host computer is very simple. It requires only two connectors, one for the analog input board (SBC 711) and one for the analog output board (SBC 724). If a teletype is to be used it should be connected to the SBC 530 adapter supplied with the system. If it is desirable to monitor the frame times with an oscilloscope, one additional conncetor is necessary to gain access to J1 on the SBC 80/20 board. The interface specifications, pin connections, and other relevent information can be gathered from the appropriate hardware reference manuals (HRM) for the SBC 711, SBC 724, and SBC 80/20 and the data sheets for the SBC 530. The following descriptions are provided as a reference for interfacing the control loading system properly.

The three SBC boards (711, 724, and 80/20) all require the same type of 50-pin PC edge connector. The following list summarizes the manufacturers and the part numbers for suitable mating connectors. The technician is advised that the connector manufacturer's numbering system may be different from Intel's and that Intel's numbering must be used regardless of the pin numbers etched on the connectors.

50-pin PC Mating Connectors

Connector Type	Vendor	Part No.
Flat Cable	3M AMP	3415-0001 2-86792-3
Soldered	AMP VIKING TI	2-583715-3 3VH25/1JV-5 H312125
Wire-wrap	TI VIKING CDC ITT	H312125 3VH25/1JND-5 VPB01E43A00A1 EC4A050A1A
Crimp	AMP	1-583717-1

6.6.1 Analog Input Interface

The analog-to-digital conversions are handled by the Intel analog input board, SBC 711. This board is capable of making channel-to-channel conversions at a 16KHZ rate with an overall accuracy of 0.05%. The board is configured to accept sixteen single-ended channels of analog signals through a 50-pin PC edge connector (J2). The pin assignments for the respective channels can be found on page 2-10 of the SBC 711 HRM. The proper methods of cabling can also be found in Figure 2.5 of that manual. Furthermore the user-option modifications made to this board can be found in the following subsections of this manual under Hardware Modifications. The following list summarizes the use of the analog input channels for the control loader.

J2 PIN	CHANNEL	VARIABLE
4	0	PDOT
8	1	PITCH
12	2	NZ
16	3	QDOT
20	4	RDOT
24	5	ROLL
28	6	NY
32	7	PPDOT
6	8	YDOT
10	9	YAW
14	10	PTRIMSIG
18	11	RTRIMSIG
	12-15	SPARE

6.6.2 Analog Output Interface

The digital-to-analog conversions are handled by the Intel analog output board, SBC 724. This board is capable of making D/A conversions with 12-bit resolution with an overall accuracy of 0.05%. The board is configured to produce four analog output channels through a 50-pin PC edge connector (J1). The pin assignments for the respective channels can be found on page 2-8 of the SBC 724 HRM. The proper methods of cabling for the outputs can be found in Figure 2.3 of that manual. Furthermore the user-option modifications made to this board can be found in the following subsection of this manual under Hardware Modifications. The following list summarizes the use of the analog output channels for the control loader.

J1 PIN	CHANNEL	VARIABLE
42	0	DAC0 (PITCH FORCE)
36	1	DAC1 (ROLL FORCE)
30	2	DAC2 (YAW FORCE)
	3	SPARE

6.6.3 Teletype Interface

Serial communication with the SBC 80/20 is conducted through a 26-pin PC edge connector (J3) on the SBC 80/20 circuit board. (See page 5-5, SBC 80/20 HRM.) However, this interface is designed for an RS232C (CRT) type of terminal. To interface a teletype (TTY) an adapter, the SBC 530, is necessary and is supplied with the control loading microcomputer. This adapter converts the RS232C signals to 20 ma current loops. It required no modification and merely plugged into J3 on the SBC 80/20. The teletype connector on the SBC 530 (J3) is a standard DB25S (female) with the following pin assignments:

PIN	FUNCTION
25	TTY TX DATA RETURN
13	TTY TX DATA
24	TTY RX DATA RETURN
12	TTY RX DATA

Thus, the Air Force must provide a teletype with a DB25P (male) connector and the above connections.

6.6.4 Frame-Time Interface

It may be desirable to monitor the frame-time information concerning the control loader microcomputer. This information is available at the SBC 80/20 connector J1 (see SBC 80/20 HRM, p. 5-3) and is summarized as follows:

PIN	BIT	PORT	FUNCTION
48	0	1	Frame-available time
46	1	1	Frame-used time
49	-	-	Ground

The SBC 80/20 has six digital I/O ports of eights bits each. The amount of frame-available time is displayed by an alternating voltage level on Bit 0 of Port 1 (pin 48, J1). The amount of frame-used time is displayed by a high voltage (5V) level at Bit 1 of Port 1 (pin 46, J1). This frame-used time is a function of the mathematical operations used in calculating the respective control forces. The pulse-width will therefore vary as the controls of the loading system are moved. Figure 6.4 illustrates the nature of these signals. Monitoring these signals has the advantage that the user can assure himself that a parameter change has not resulted in any framing errors and that sufficient frame time remains for further changes.

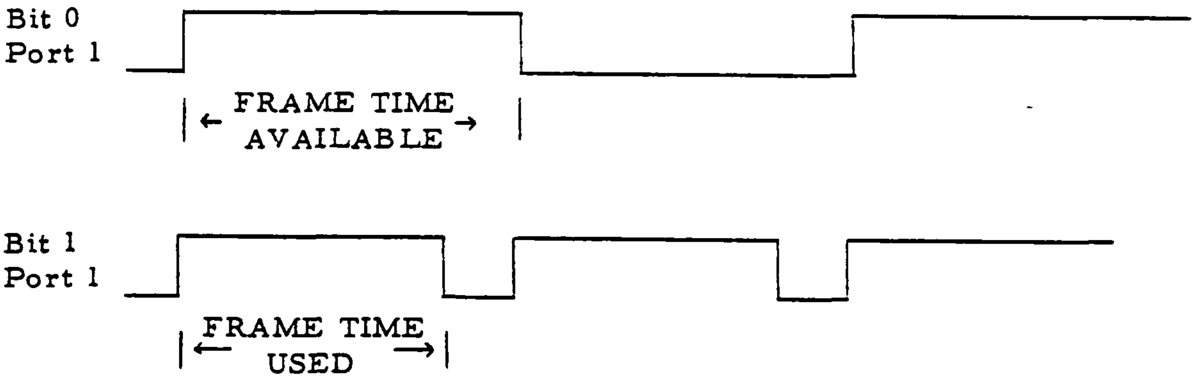


Figure 6.4 Frame Time Pulses

6.7 SYSTEM OPERATION

The microcomputer carries with it a great deal of simplicity. Its operation does not require a high degree of skill in the digital sciences or computer technology. The following paragraphs relate the tasks which the system operator must accomplish to initiate the control loading simulator or any of the other options.

6.7.1 Interfacing

The minimum "black box" configuration requires that the analog inputs be connected to the control loader and host computer as described in Section 6.6.1. In addition, the analog outputs must be connected to the control loader as described in Section 6.6.2. The microcomputer is therefore a stand alone device as illustrated in Figure 1.1. If the operator desires to change a parameter or execute any of the self-test features built into the microcomputer, a teletype must be connected to the SBC 530 TTY adapter as described in Section 6.6.3.

6.7.2 Power Up and Simulator Execution

In the minimum "black box" configuration described above, the only required operation is to apply power to the SBC 80/20 by depressing the "power on" switch on the left side of the front panel. The simulator is automatically entered and the system is ready for immediate use with the control loading system.

6.7.3 Optional Features

The system operator may elect to use some of the options associated with the microcomputer. He may elect to reset both the hardware and software through the use of the RESET button on the right side of the front panel. If he connects a teletype to the system he may change parameters and/or initiate one of the self-test features.

6.7.3.1 Reset

The CPU is equipped with a "power-on reset" feature such that .5 second reset pulse is produced when power is first applied. If the operator depresses the reset button on the right side of the front panel the same action will be taken as when the power is first applied, except that the reset level remains active as long as the button is held.

AD-A100 429 DAYTON UNIV OH SCHOOL OF ENGINEERING F/G 9/2
COMPUTER PROGRAM DOCUMENTATION FOR MICROCOMPUTER IMPLEMENTATION--ETC(U)
OCT 78 F G ALBERS, J CROUCH F33657-77-C-0477
UNCLASSIFIED UDR-TR-78-98 NL

2 OF 2
AD A100429

END
DATE
FILMED
7-81
DTIC

CAUTION

When the reset button is depressed the analog output board commands all of the DAC's to go to -10v. Caution must be taken to disconnect or otherwise disable the control loader so that the servoes are not commanded to go to the negative limit. A hardware modification can be made to correct this situation if necessary.

6.7.3.2 Parameter Changes

The operator may enter parameter changes by first depressing a key (any key) on the teletype console while the simulator is active. This action will cause the simulator loop to be temporarily interrupted and the loading on the stick and rudder will be zeroed (all DAC's = 0v). The operator is presented with the query

CHANGE PARAMETER? (Y or N)

If he responds in the negative (N) the system will assume he wishes to execute a self-test program. If he responds in the positive (Y) the system will pursue this request further by typing

PARAMETER NUMBER?

The processor is now waiting for the number of the parameter to be changed (see Section 6.1 and 5.4.4.1). The parameters and the associated numbers are listed in Table 5.1. If the operator depresses the carriage return without inputting a parameter number the processor immediately returns to the simulator loop.

CAUTION

The stick and rudders should be placed in a position close to neutral to prevent rapid build-up of forces.

If, on the other hand, the operator selects one of the parameters, the processor will display that parameter's present value, e.g.

PRESENT VALUE = 23522

The system then asks for a new value by typing

NEW VALUE?

If the operator inputs a carriage return without first inputting a value, that parameter remains unchanged and control passes back to the "parameter number" question. This feature allows the operator to interrogate parameters without changing them. Of course if he does supply a valid integer value (≤65535), that parameter will be changed appropriately. Once again, the parameter change routine is terminated and the simulation is resumed by typing a carriage return immediately after being asked for a parameter number.

6.7.3.3 Self-Test Programs

If the operator desires to execute a calibration or test program while the simulator is operating he must first depress a key (any key) on the teletype console. This action terminates the simulator loop and causes the system to ask the operator

CHANGE PARAMETER? (Y OR N)

The response must be negative (N) for which the system will enter the executive program as evidenced by the console message:

CONTROL LOADER EXECUTIVE
WHICH PROGRAM?

The operator may now select any of the self test programs (0 through 4) by typing the appropriate number. (See Section 5.2.7 for a complete description of these programs.) At the completion of a program control passes back to the executive. However, if program 5 is selected, control passes back to the simulator; that is, all simulator parameters are reinitialized and the simulator loop is entered.

APPENDIX A
MODULE FLOW CHARTS

This section contains the detailed module flow charts referenced in Section 5. Each flow chart corresponds to the procedure of the same name as described in that section.

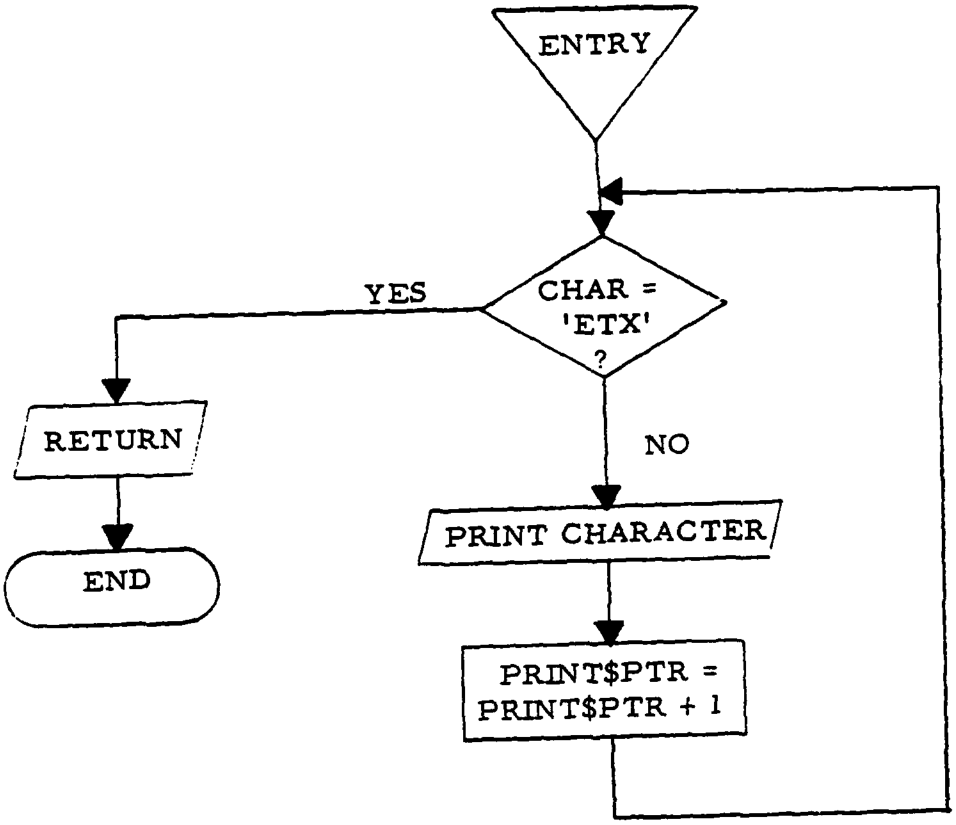


Figure A. 1
PRINT PROCEDURE

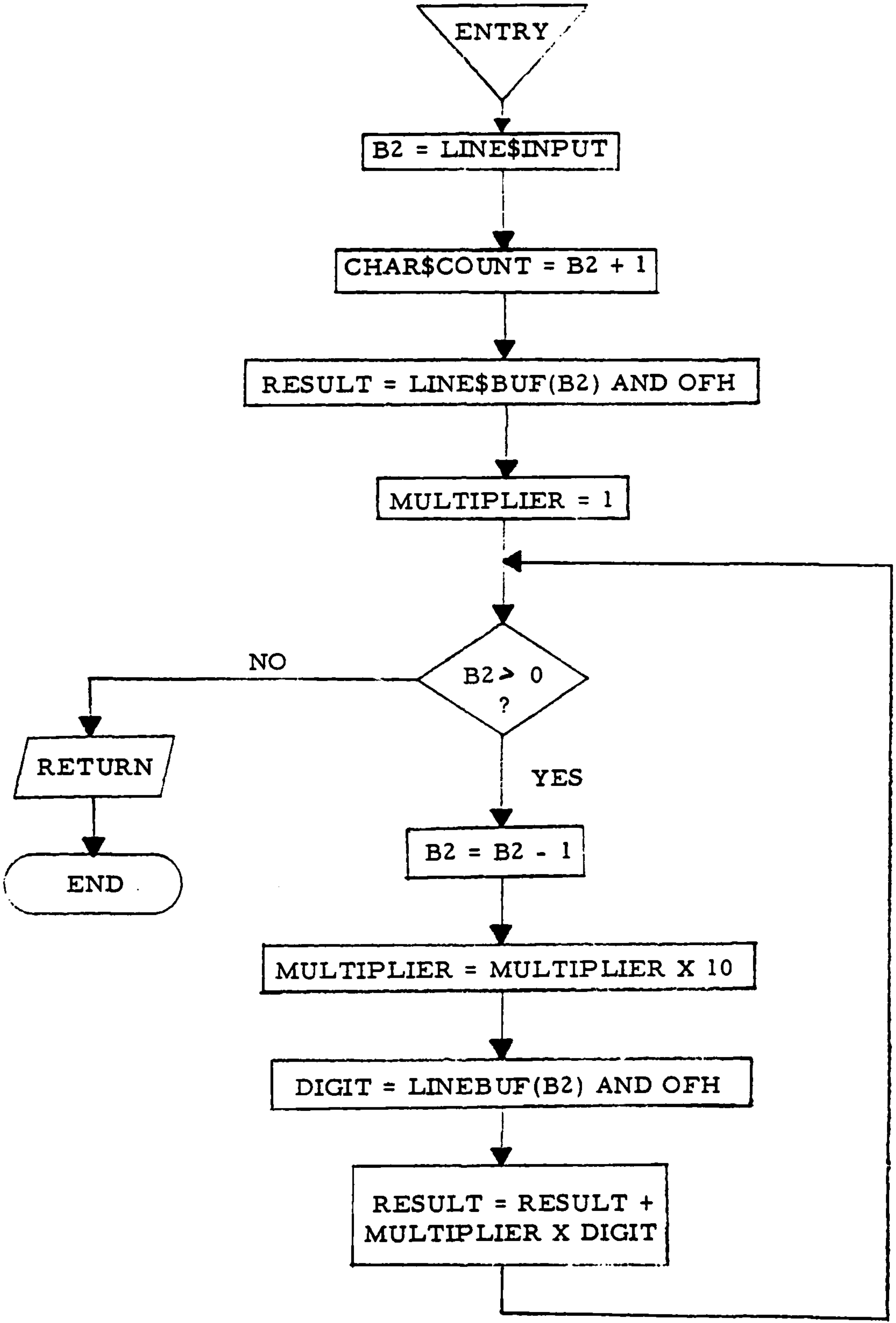

ENTRY
B2 = LINE$INPUT
CHAR$COUNT = B2 + 1
RESULT = LINE$BUF(B2) AND OFH
MULTIPLIER = 1
B2 > 0 ?
NO
RETURN
END
YES
B2 = B2 - 1
MULTIPLIER = MULTIPLIER X 10
DIGIT = LINEBUF(B2) AND OFH
RESULT = RESULT + MULTIPLIER X DIGIT

Figure A. 2
ASCII$TO$FIX PROCEDURE

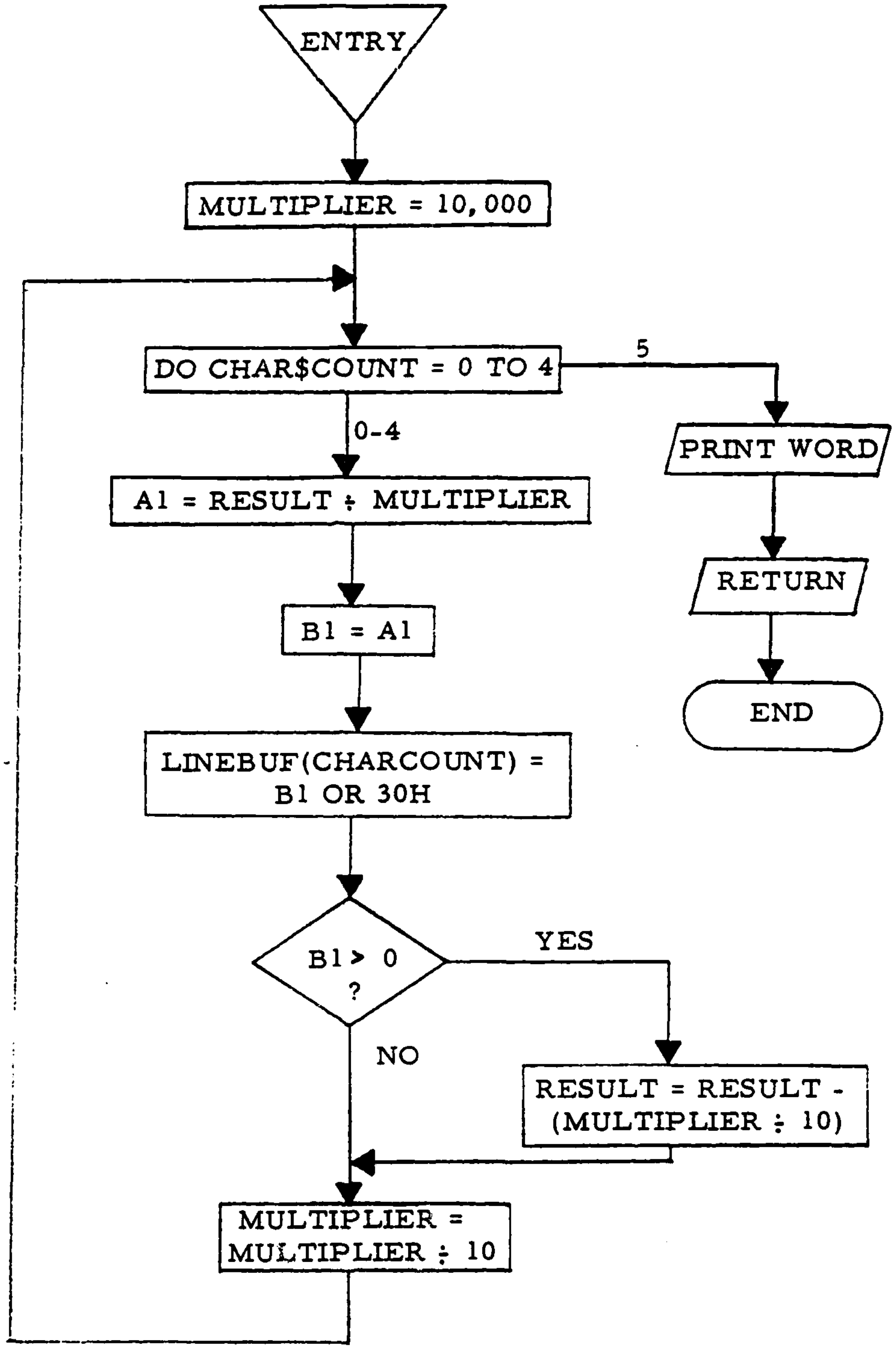


Figure A.3
FIX$TO$ASCII PROCEDURE

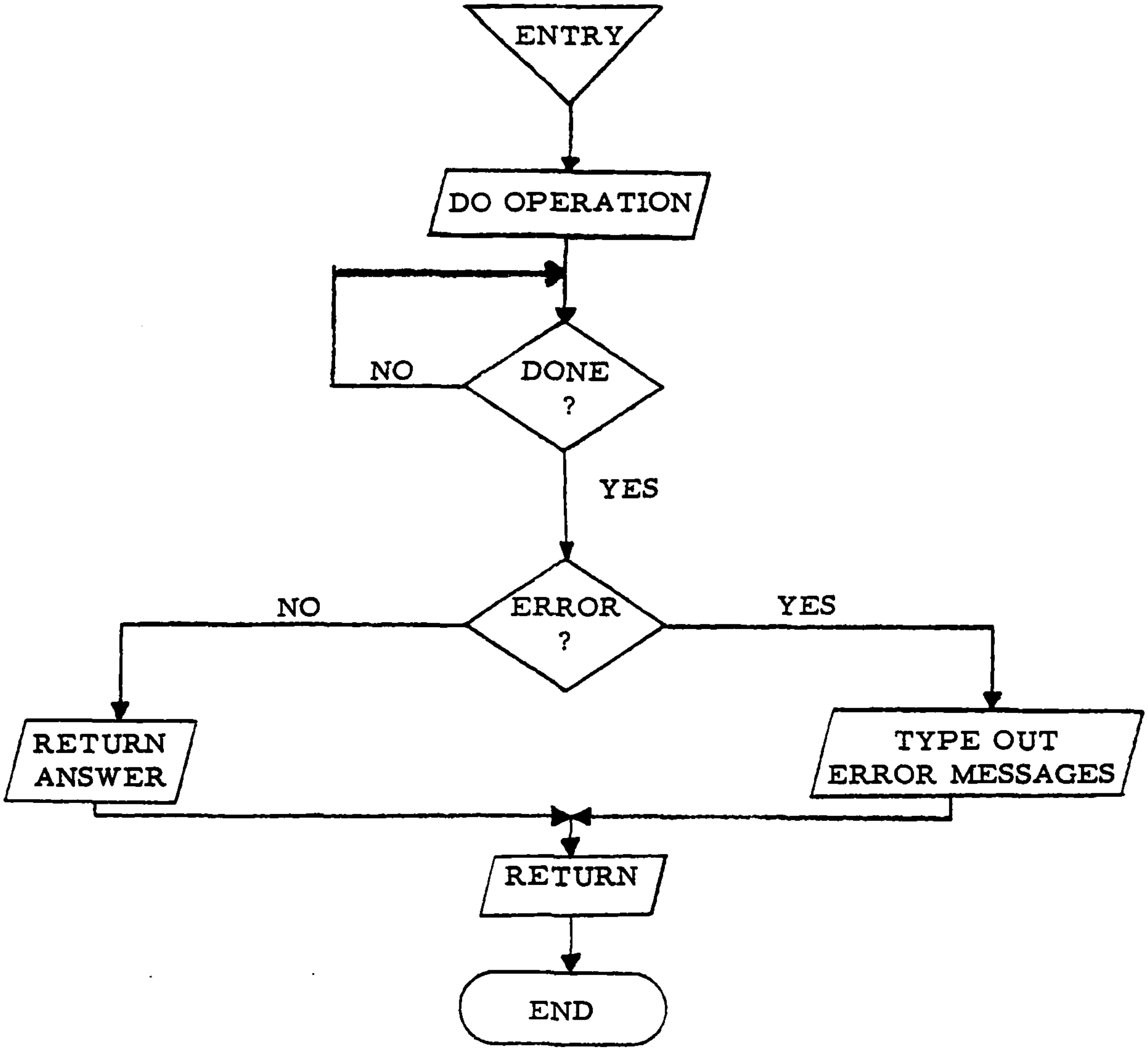


Figure A.4
PERFORM PROCEDURE

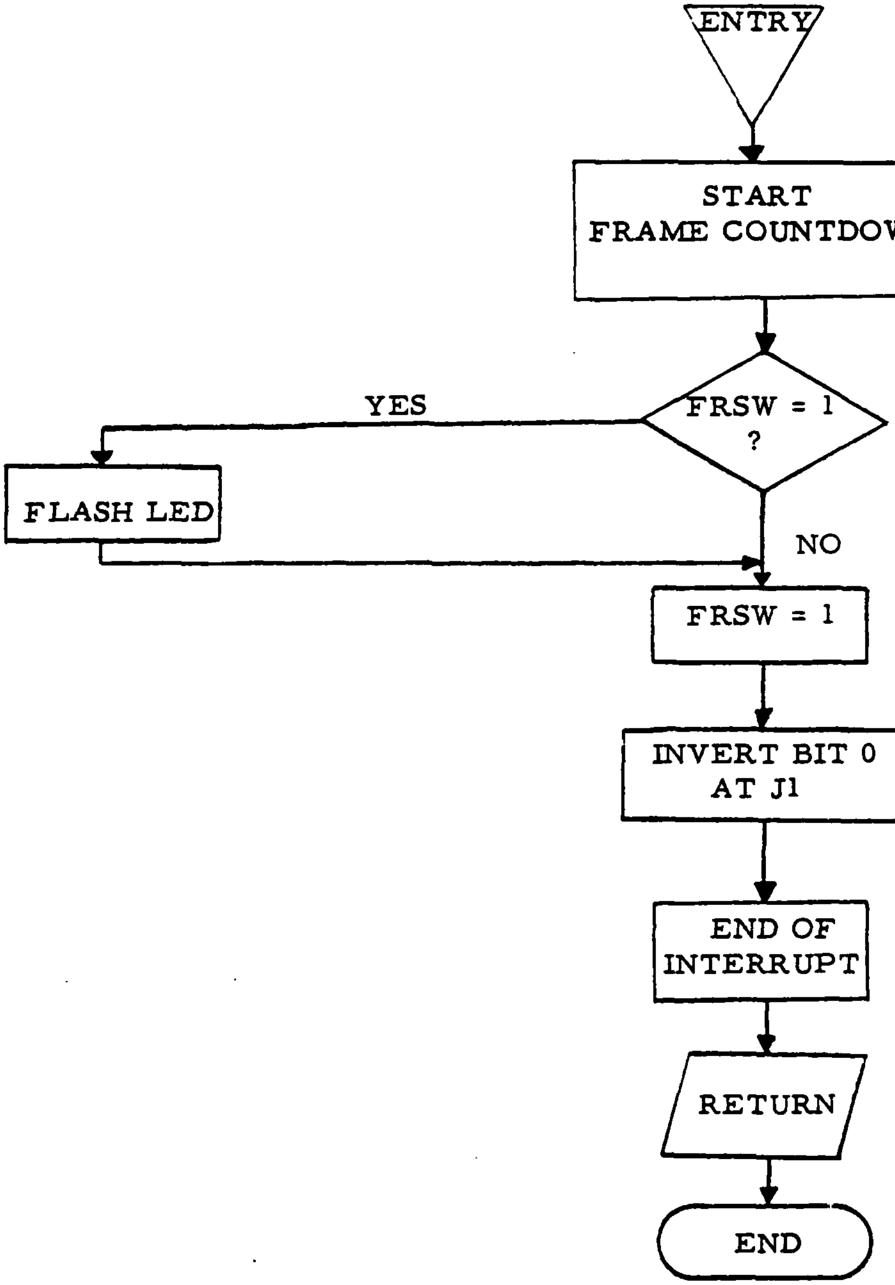


Figure A.5
FRAME PROCEDURE

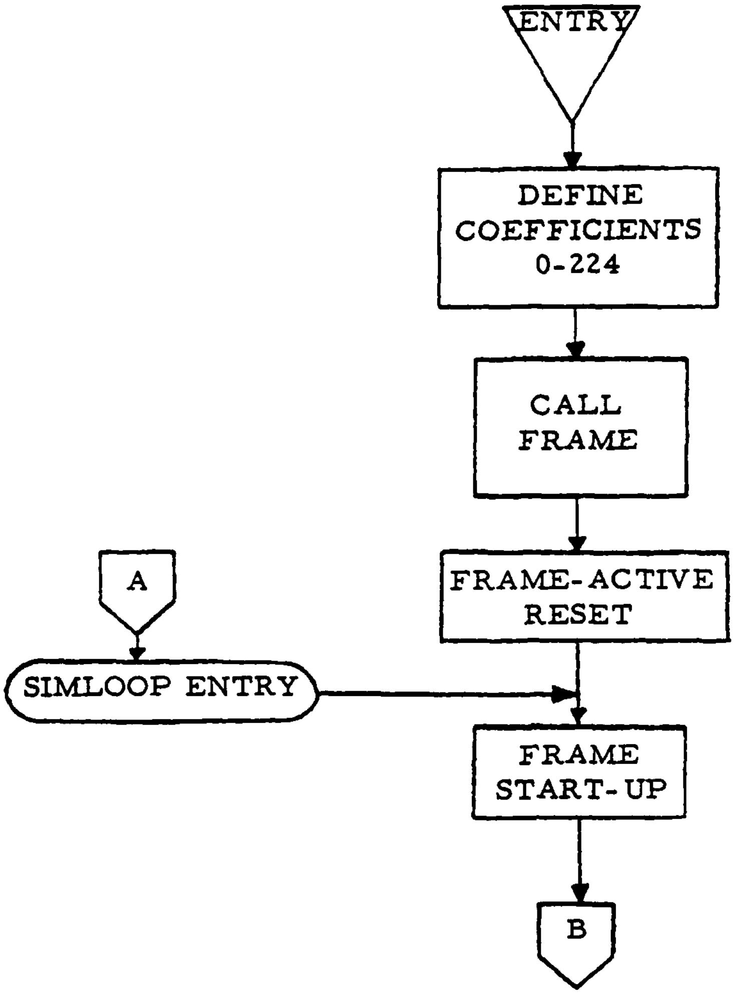


Figure A. 6. 1
SIMULATOR PROCEDURE

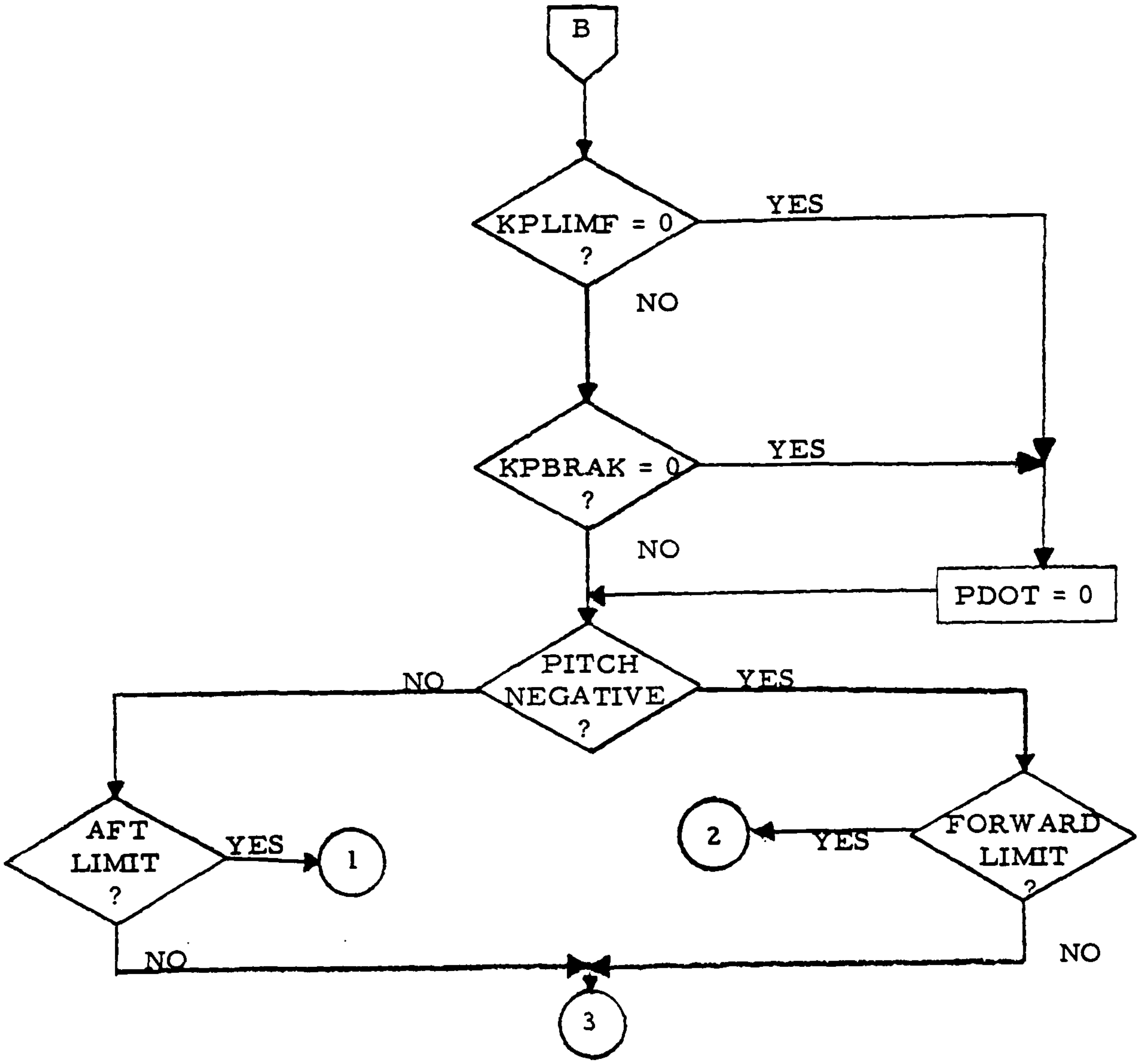

B
KPLIMF = 0 ?
YES
NO
KPBRAK = 0 ?
YES
NO
PDOT = 0
PITCH NEGATIVE ?
NO
YES
AFT LIMIT ?
YES
1
NO
FORWARD LIMIT ?
YES
2
NO
3

Figure A.6.2

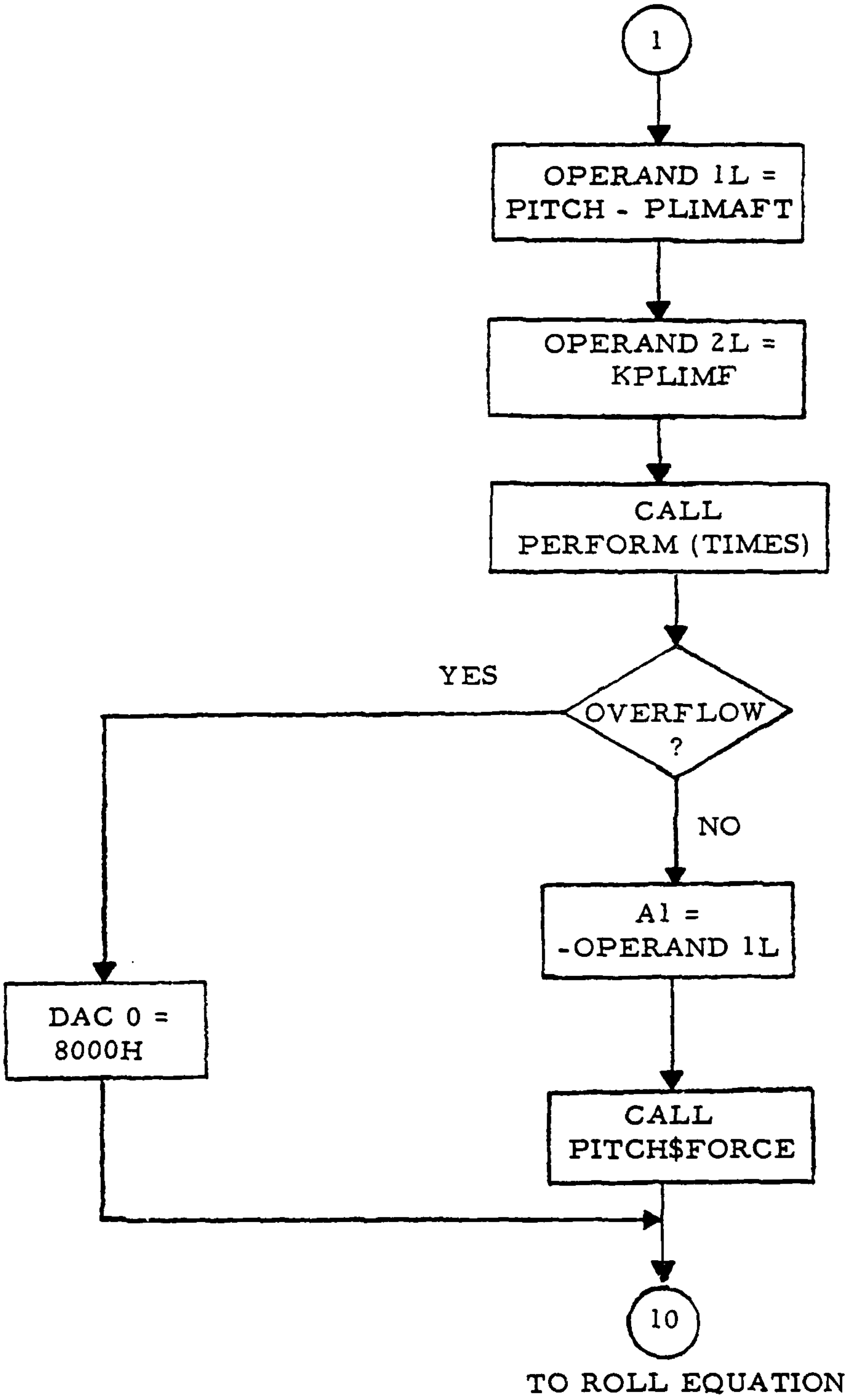

1
OPERAND 1L =
PITCH - PLIMAFT
OPERAND 2L =
KPLIMF
CALL
PERFORM (TIMES)
YES
OVERFLOW
?
NO
A1 =
-OPERAND 1L
DAC 0 =
8000H
CALL
PITCH$FORCE
10
TO ROLL EQUATION

Figure A.6.3

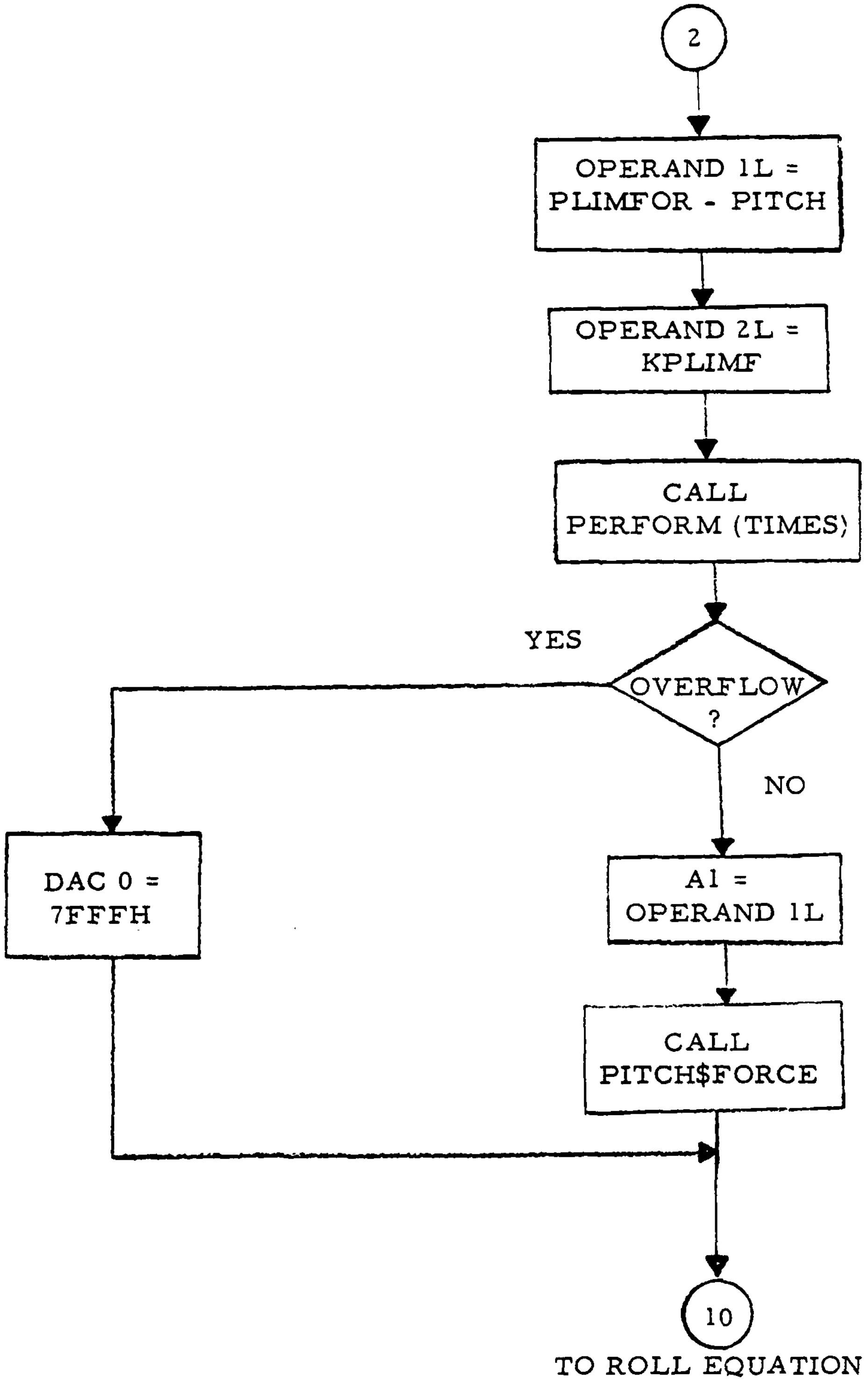

2
OPERAND 1L = PLIMFOR - PITCH
OPERAND 2L = KPLIMF
CALL PERFORM (TIMES)
YES
OVERFLOW ?
NO
DAC 0 = 7FFFH
A1 = OPERAND 1L
CALL PITCH$FORCE
10
TO ROLL EQUATION

Figure A. 6. 4

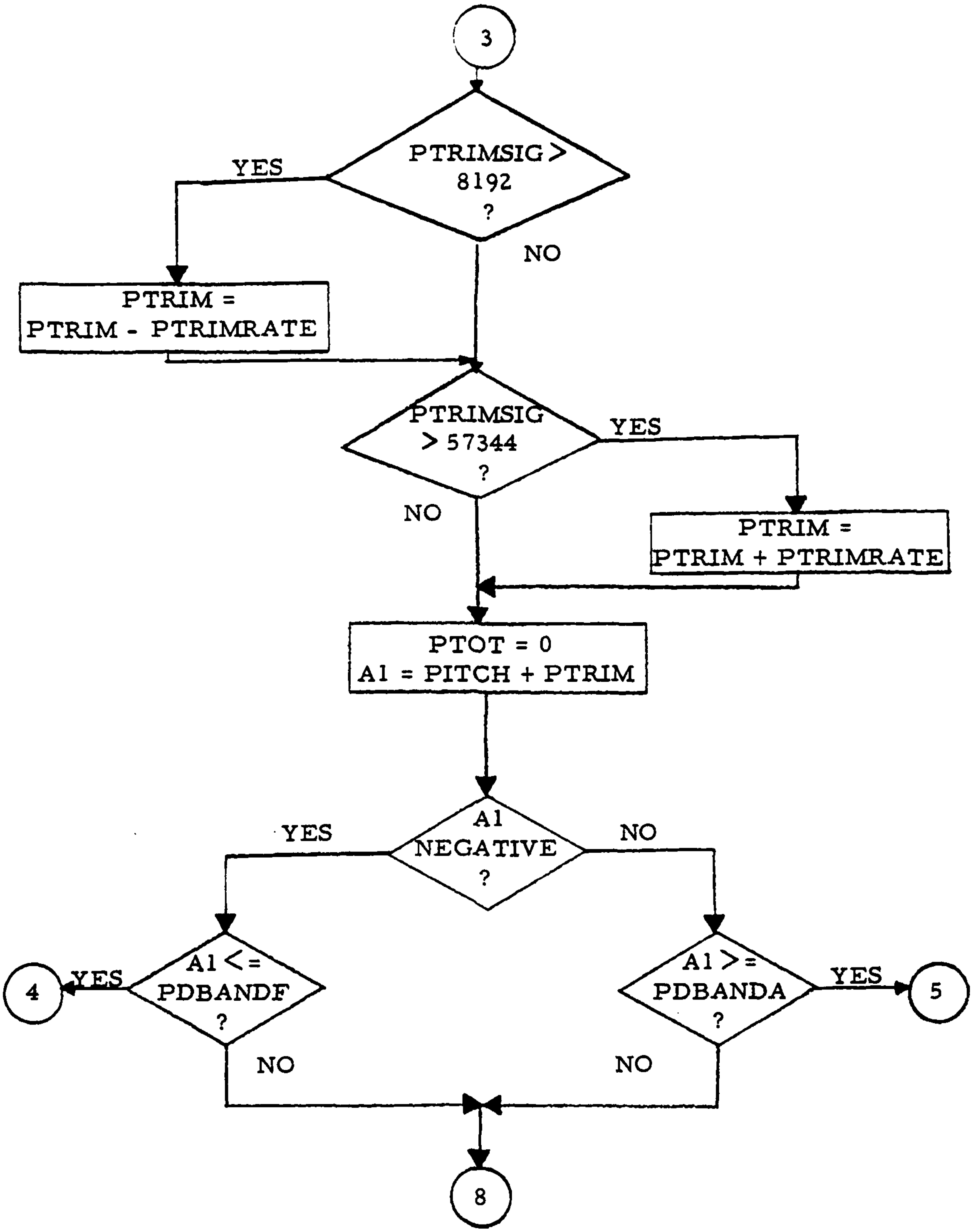

3
PTRIMSIG > 8192 ?
YES
NO
PTRIM = PTRIM - PTRIMRATE
PTRIMSIG > 57344 ?
YES
NO
PTRIM = PTRIM + PTRIMRATE
PTOT = 0
A1 = PITCH + PTRIM
A1 NEGATIVE ?
YES
NO
A1 <= PDBANDF ?
YES
4
NO
A1 >= PDBANDA ?
YES
5
NO
8

Figure A.6.5

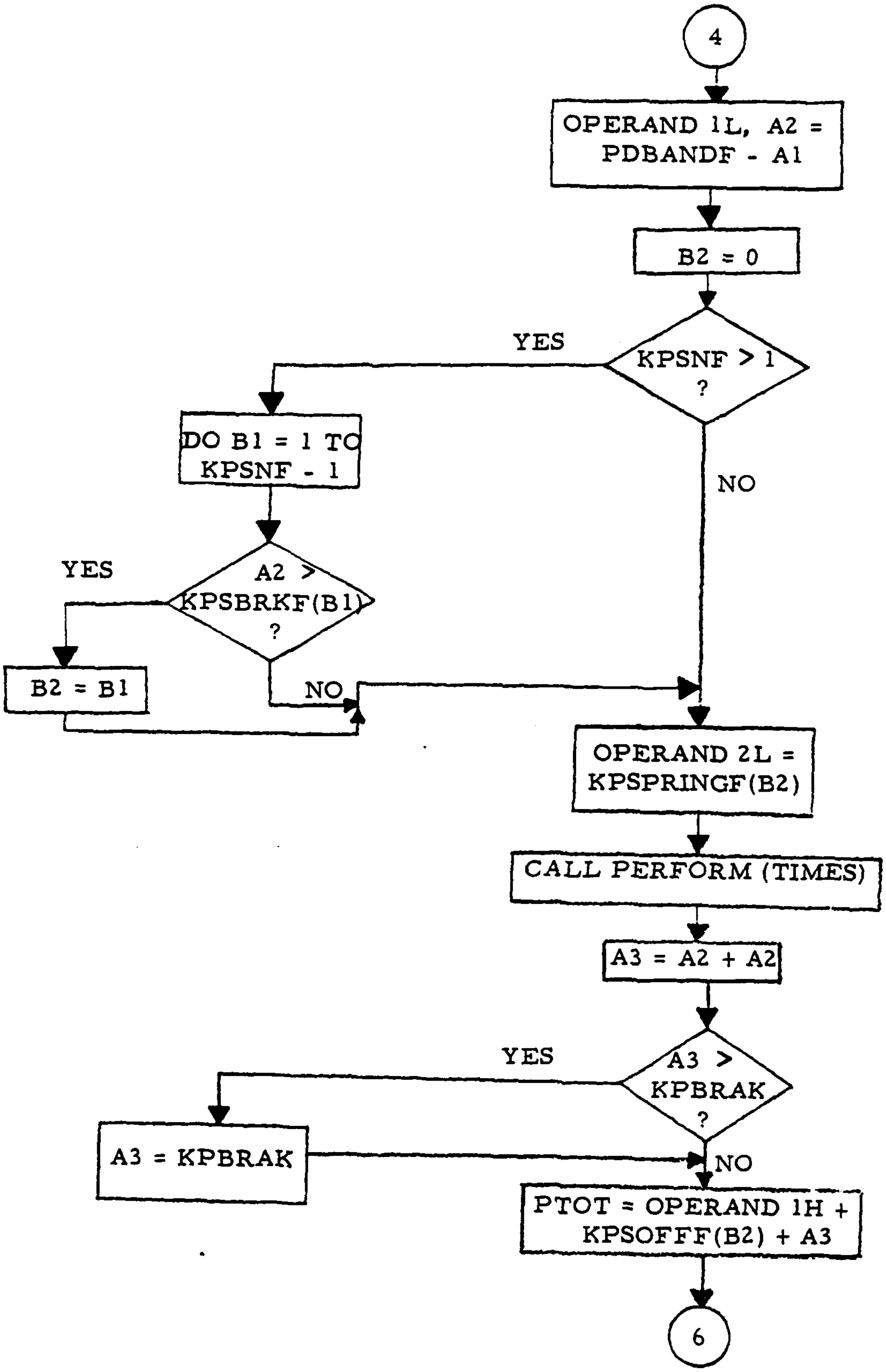

4
OPERAND 1L, A2 = PDBANDF - A1
B2 = 0
KPSNF > 1 ?
YES
NO
DO B1 = 1 TO KPSNF - 1
A2 > KPSBRKF(B1) ?
YES
B2 = B1
NO
OPERAND 2L = KPSPRINGF(B2)
CALL PERFORM (TIMES)
A3 = A2 + A2
A3 > KPBRAK ?
YES
A3 = KPBRAK
NO
PTOT = OPERAND 1H + KPSOFFF(B2) + A3
6

Figure A.6.6

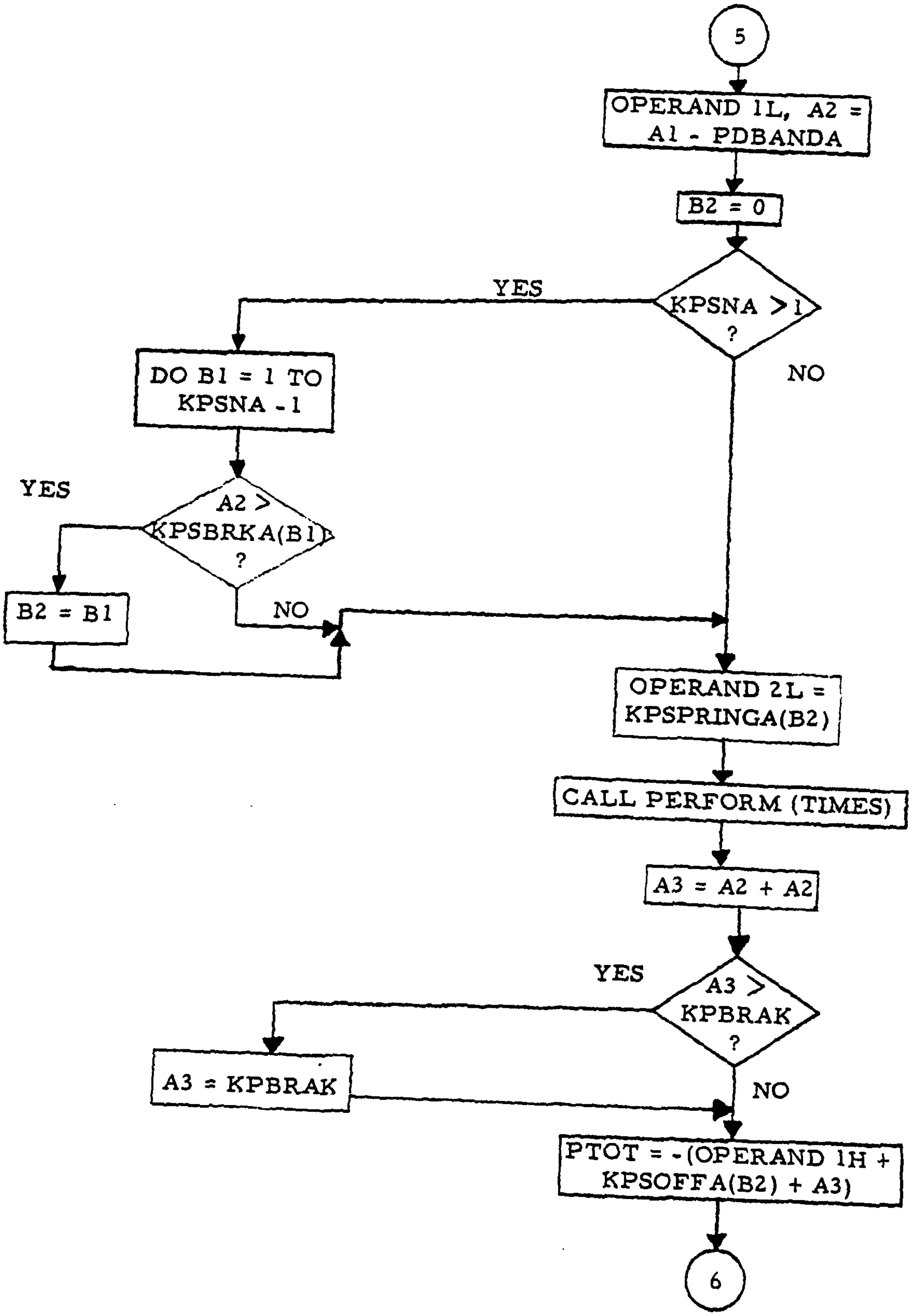

5
OPERAND 1L, A2 = A1 - PDBANDA
B2 = 0
KPSNA > 1 ?
YES
NO
DO B1 = 1 TO KPSNA - 1
YES
A2 > KPSBRKA(B1) ?
NO
B2 = B1
OPERAND 2L = KPSPRINGA(B2)
CALL PERFORM (TIMES)
A3 = A2 + A2
YES
A3 > KPBRAK ?
A3 = KPBRAK
NO
PTOT = -(OPERAND 1H + KPSOFFA(B2) + A3)
6

Figure A.6.7

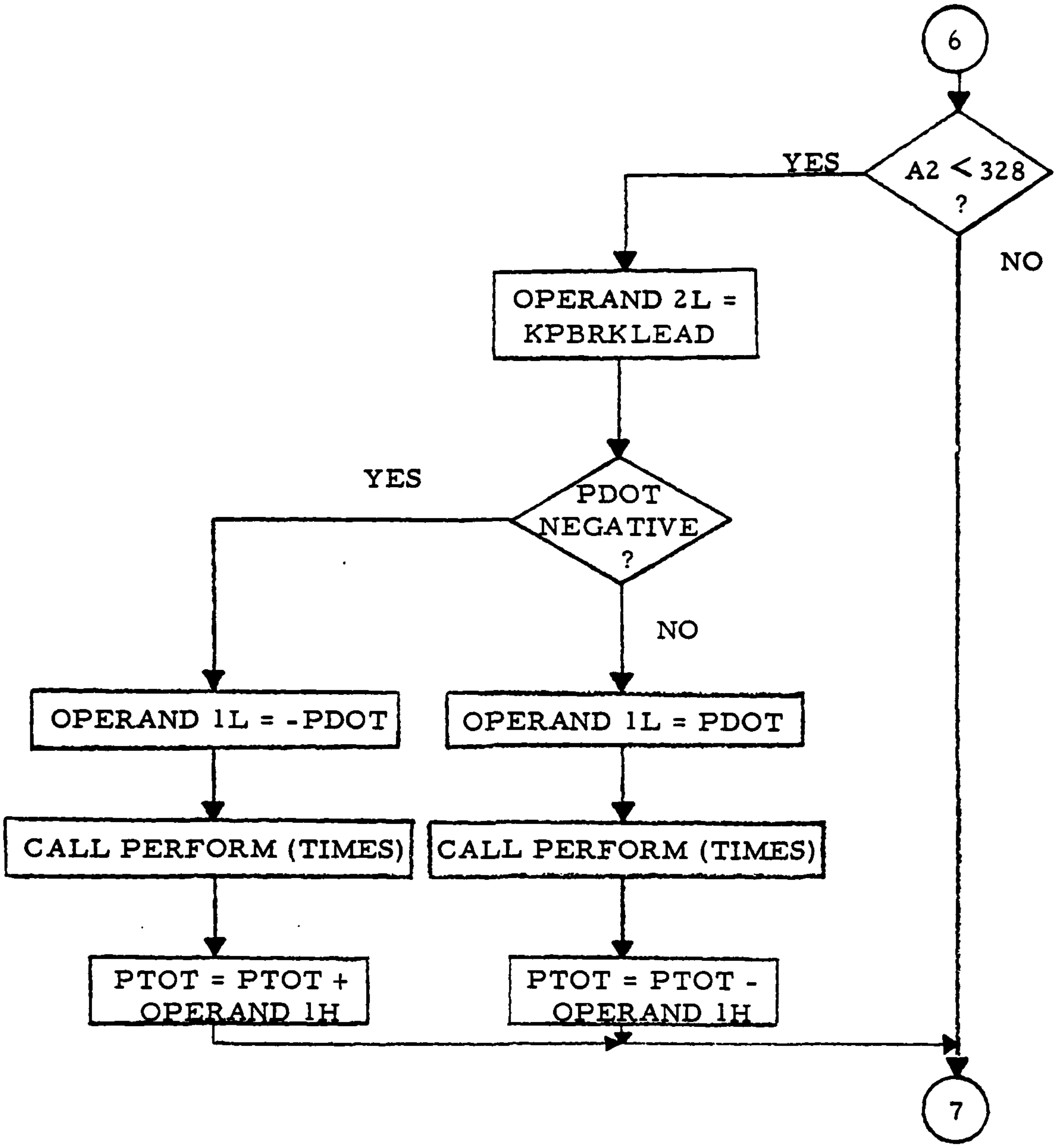

6
A2 < 328 ?
YES
NO
OPERAND 2L = KPBRKLEAD
PDOT NEGATIVE ?
YES
NO
OPERAND 1L = -PDOT
OPERAND 1L = PDOT
CALL PERFORM (TIMES)
CALL PERFORM (TIMES)
PTOT = PTOT + OPERAND 1H
PTOT = PTOT - OPERAND 1H
7

Figure A. 6. 8

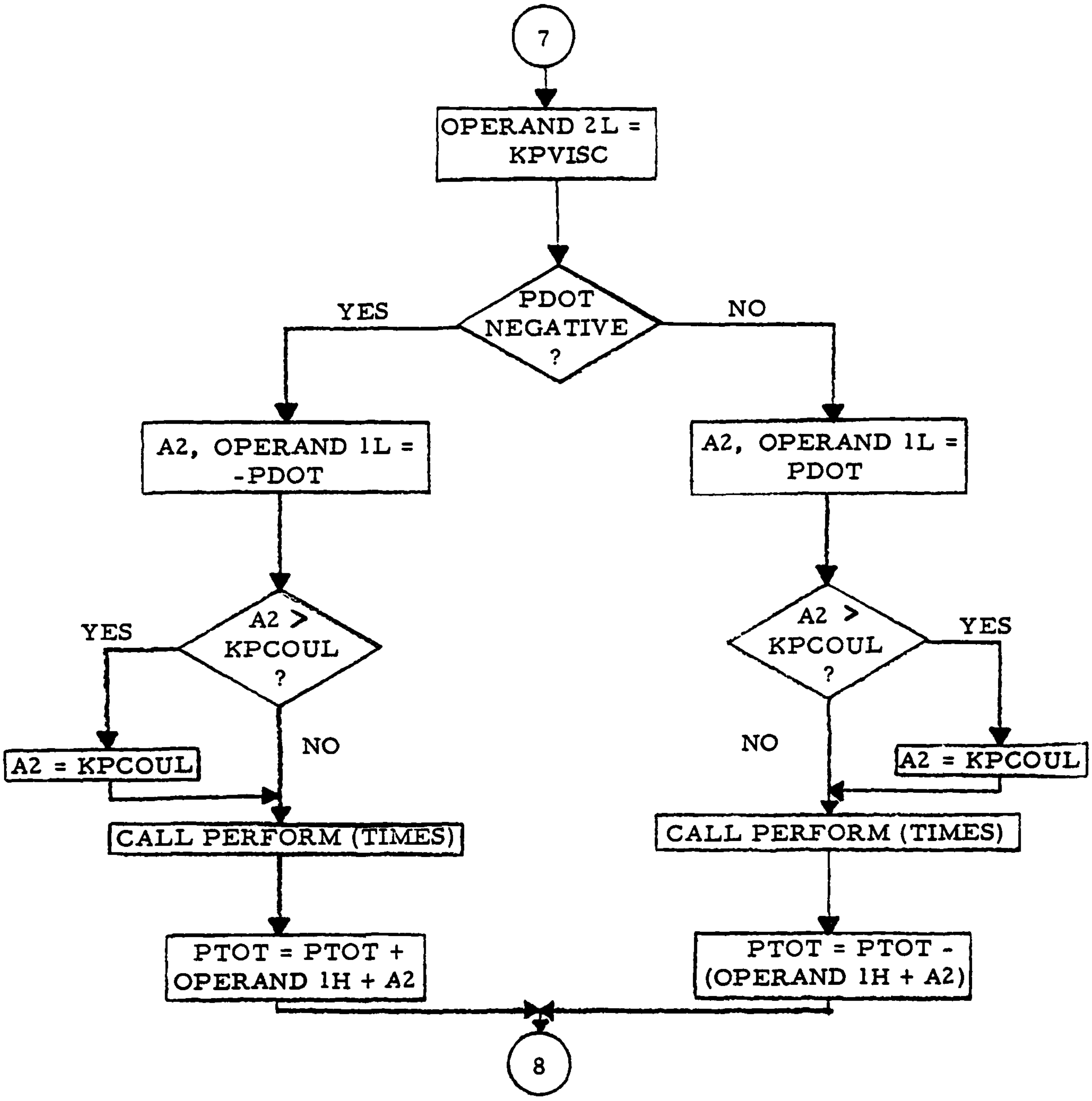

7
OPERAND 2L = KPVISC
PDOT NEGATIVE ?
YES
NO
A2, OPERAND 1L = -PDOT
A2, OPERAND 1L = PDOT
A2 > KPCOUL ?
YES
NO
A2 = KPCOUL
A2 > KPCOUL ?
YES
NO
A2 = KPCOUL
CALL PERFORM (TIMES)
CALL PERFORM (TIMES)
PTOT = PTOT + OPERAND 1H + A2
PTOT = PTOT - (OPERAND 1H + A2)
8

Figure A.6.9

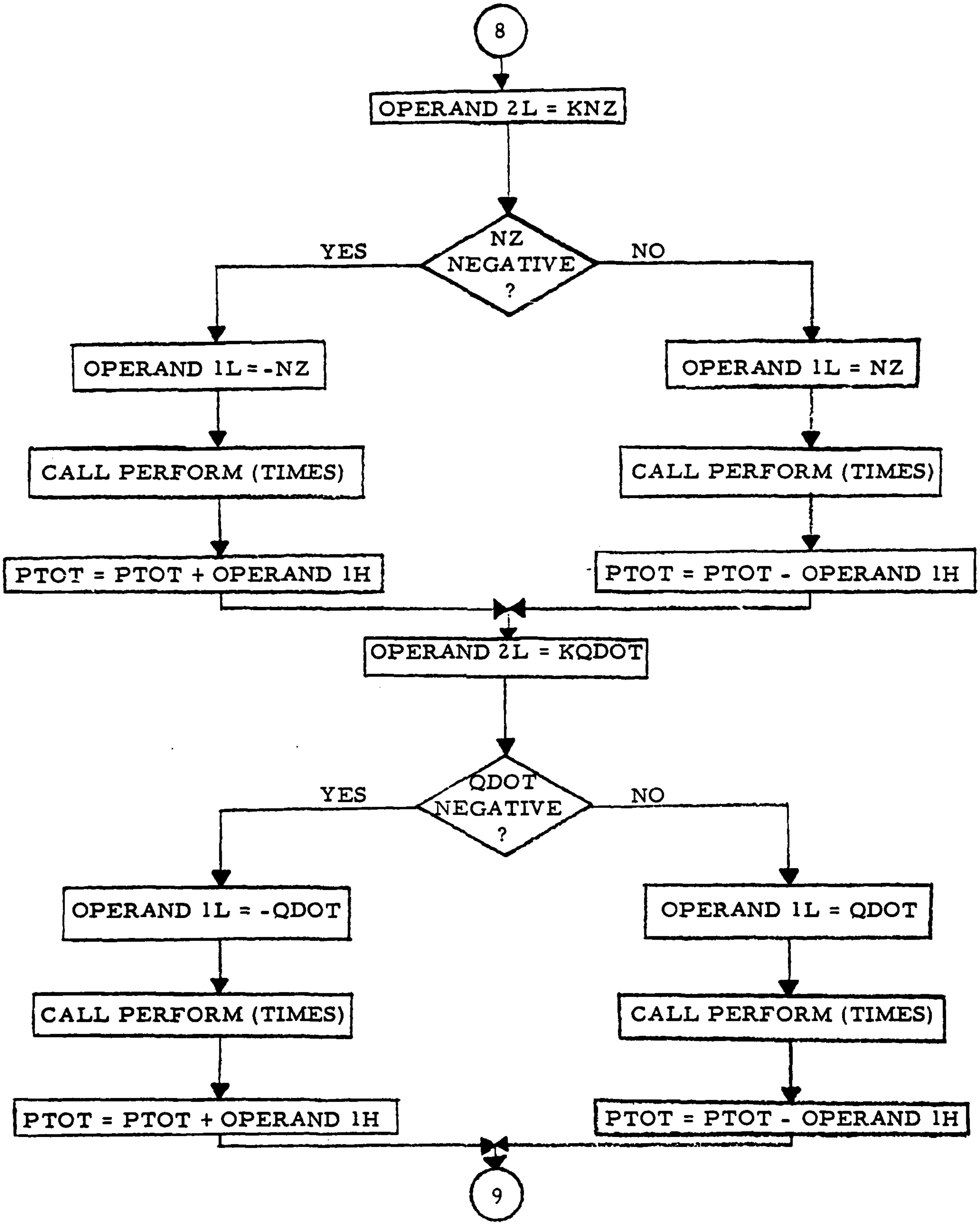

8
OPERAND 2L = KNZ
NZ NEGATIVE ?
YES
NO
OPERAND 1L = -NZ
OPERAND 1L = NZ
CALL PERFORM (TIMES)
CALL PERFORM (TIMES)
PTOT = PTOT + OPERAND 1H
PTOT = PTOT - OPERAND 1H
OPERAND 2L = KQDOT
QDOT NEGATIVE ?
YES
NO
OPERAND 1L = -QDOT
OPERAND 1L = QDOT
CALL PERFORM (TIMES)
CALL PERFORM (TIMES)
PTOT = PTOT + OPERAND 1H
PTOT = PTOT - OPERAND 1H
9

Figure A. 6. 10

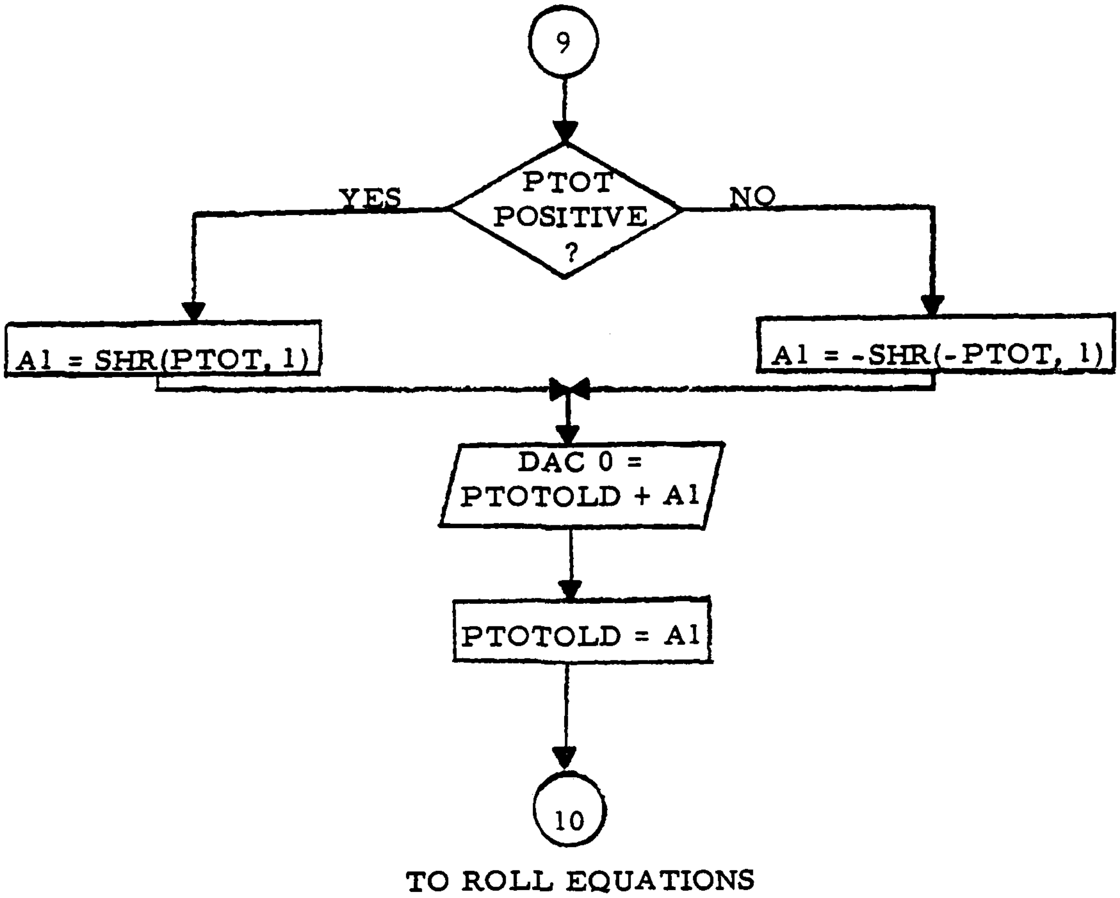
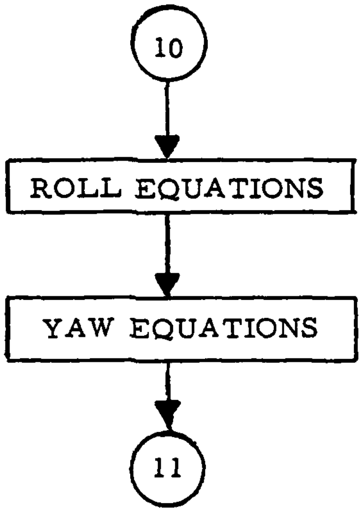

9
PTOT POSITIVE ?
YES
NO
A1 = SHR(PTOT, 1)
A1 = -SHR(-PTOT, 1)
DAC 0 = PTOTOLD + A1
PTOTOLD = A1
10
TO ROLL EQUATIONS

Figure A.6.11



Note: Roll and Yaw Equations are processed identical to the Pitch equations.

Figure A.6.12

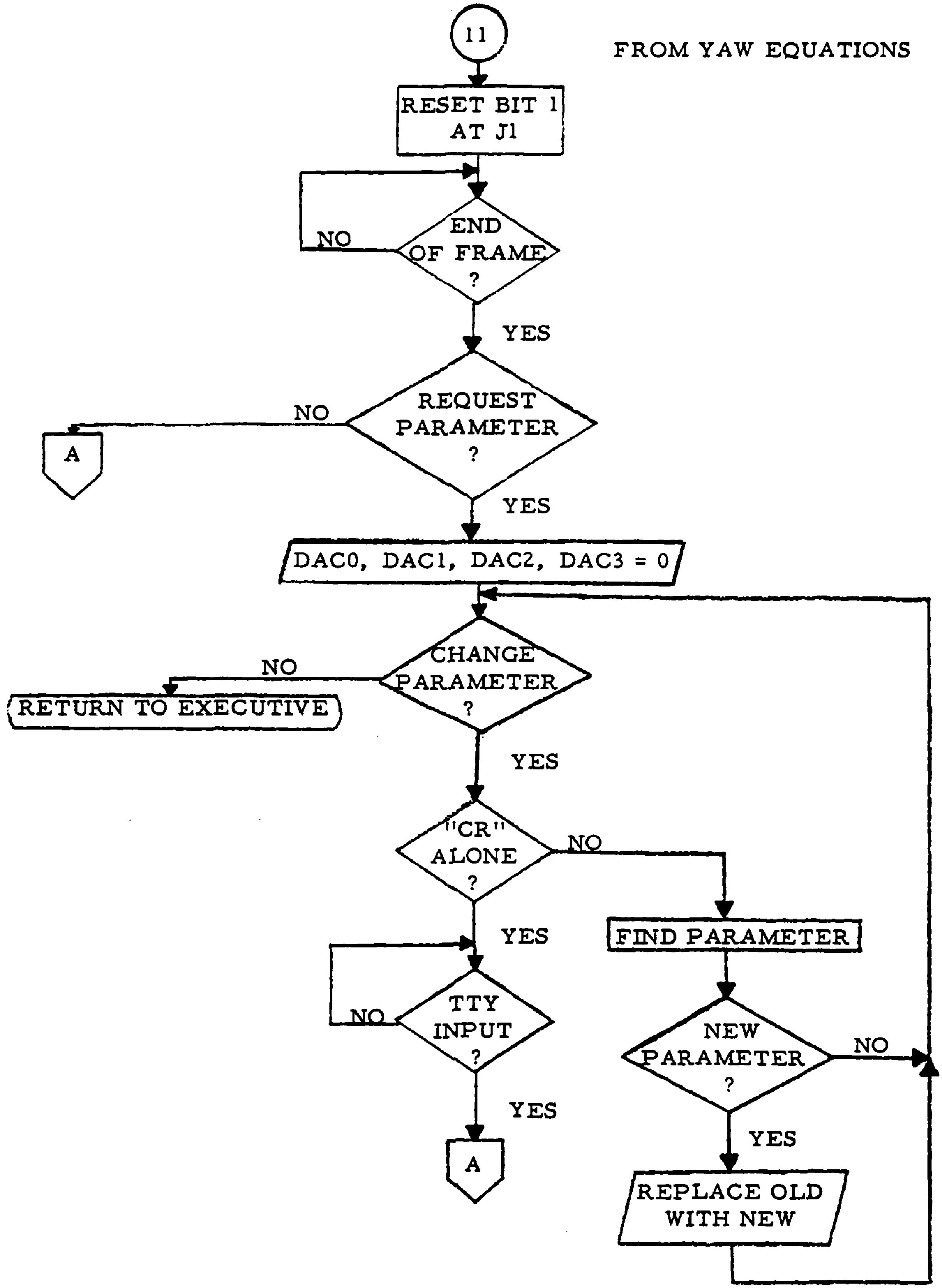

11
FROM YAW EQUATIONS
RESET BIT 1 AT J1
END OF FRAME ?
NO
YES
REQUEST PARAMETER ?
NO
A
YES
DAC0, DAC1, DAC2, DAC3 = 0
CHANGE PARAMETER ?
NO
RETURN TO EXECUTIVE
YES
"CR" ALONE ?
NO
FIND PARAMETER
YES
TTY INPUT ?
NO
YES
A
NEW PARAMETER ?
NO
YES
REPLACE OLD WITH NEW

Figure A. 6. 13

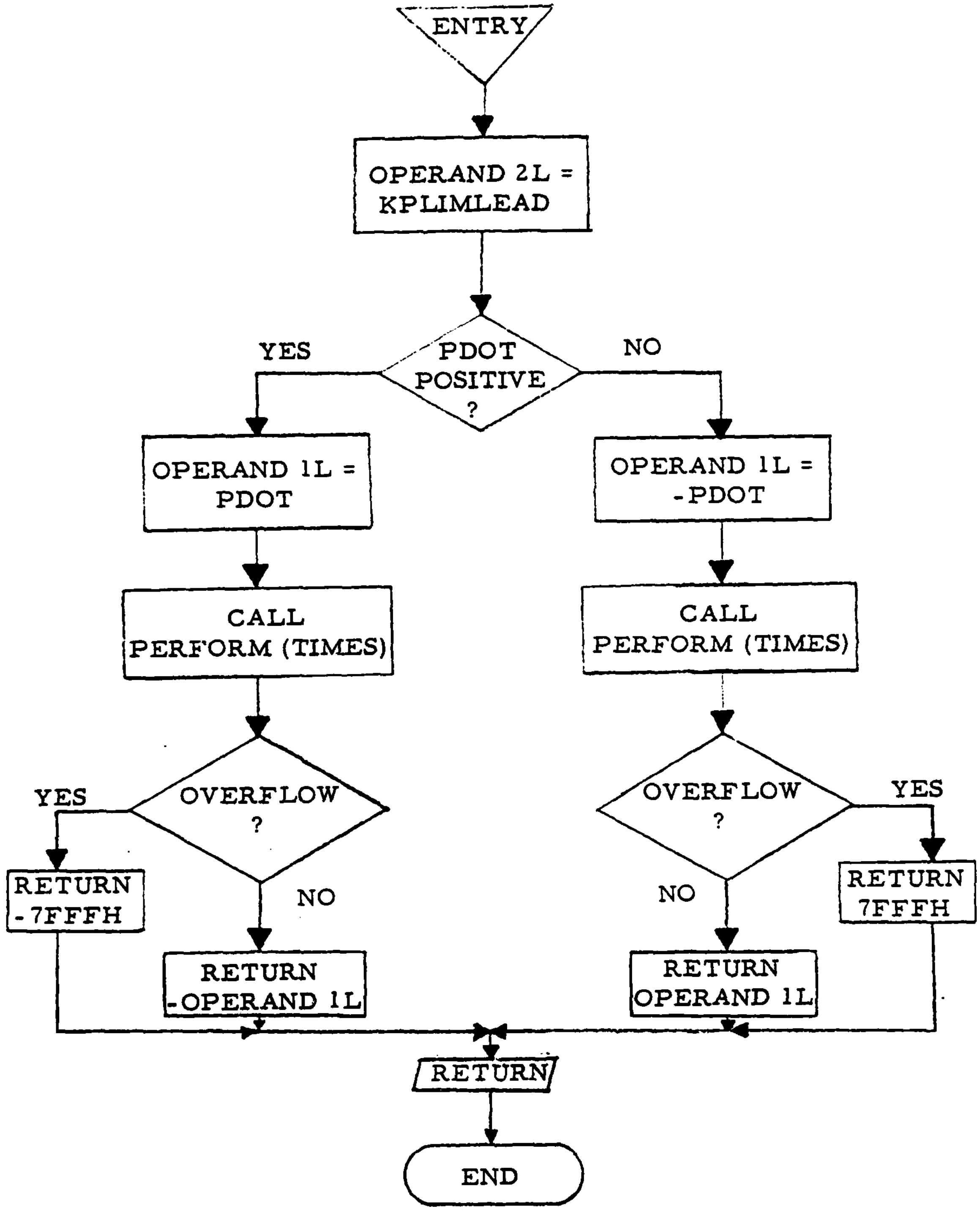


Figure A. 7
PLEAD PROCEDURE

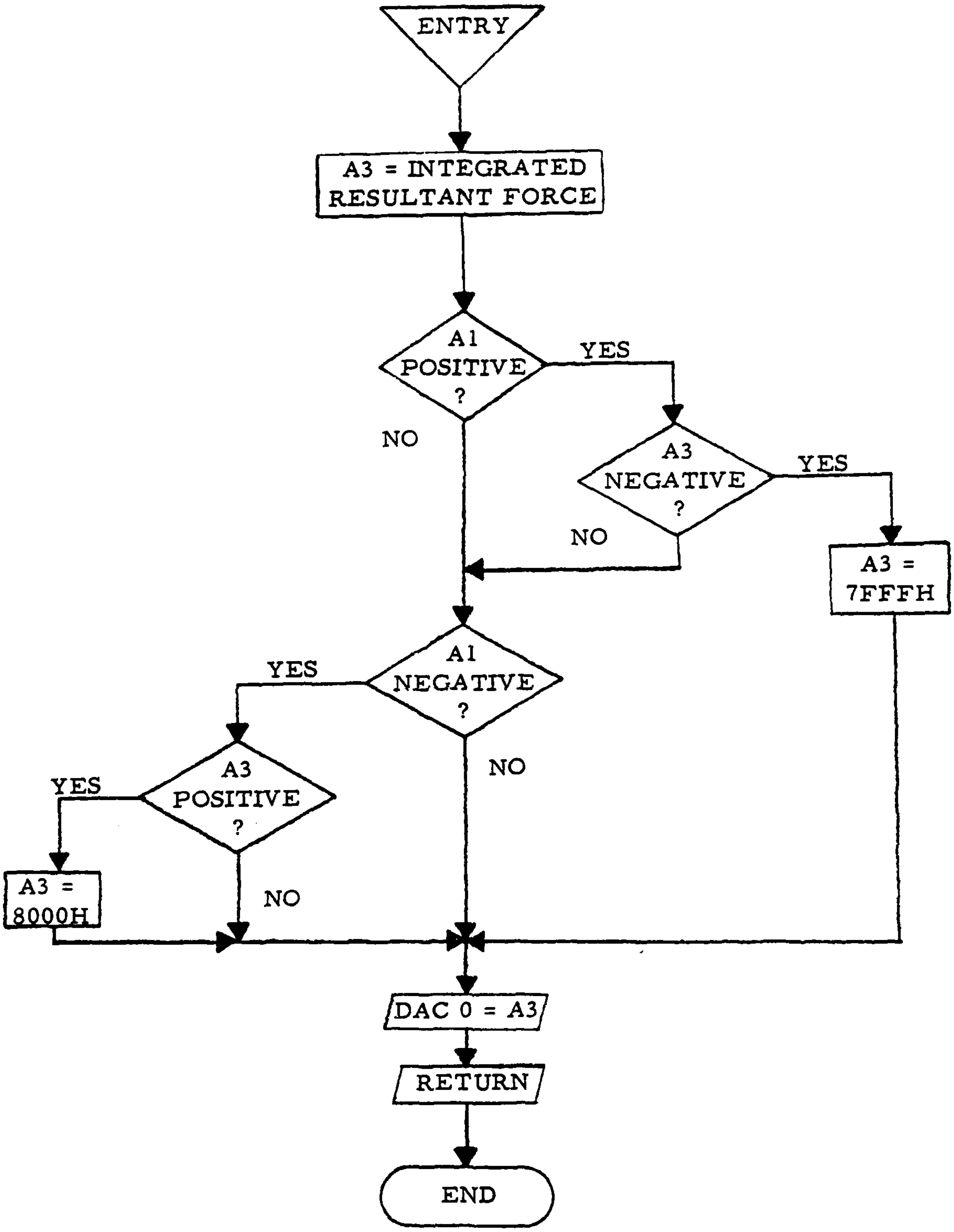


Figure A. 8
PITCH$FORCE PROCEDURE